ComponentOne

DataGrid for Silverlight

Corporate Headquarters
ComponentOne LLC
201 South Highland Avenue
3rd Floor
Pittsburgh, PA 15206 · USA

Internet: info@ComponentOne.com

Web site: http://www.componentone.com

Sales

E-mail: sales@componentone.com

Telephone: 1.800.858.2739 or 1.412.681.4343 (Pittsburgh, PA USA Office)

This manual was produced using ComponentOne Doc-To-Help™.

Table of Contents

ComponentOne DataGrid for Silverlight Overview

Add advanced data visualization to your Silverlight applications with **ComponentOne DataGrid™ for Silverlight**. The robust data-bound **C1DataGrid** control makes it easy to display, edit, and analyze tabular data in Silverlight applications.

Getting Started

Get started with the following topics:

- Key Features (page 45)

- Quick Start (page 46)

- Task-Basked Help (page 144)

Silverlight**SilverlightSilverlightSilverlightSilverlightSilverlightSilverlightSilverlight**Silverlight**SilverlightSilverlight**Installing DataGrid for Silverlight

The following sections provide helpful information on installing **ComponentOne DataGrid for Silverlight**.

DataGrid for Silverlight Setup Files

The **ComponentOne Studio for Silverlight** installation program will create the following directory: **C:\Program Files\ComponentOne\Studio for Silverlight 4.0**. This directory contains the following subdirectories:

Bin	Contains copies of ComponentOne binaries (DLLs, EXEs, design-time assemblies).
Help	Contains documentation for all Studio components and other useful resources including XAML files.

Samples

Samples for the product are installed in the **ComponentOne Samples** folder by default. The path of the ComponentOne Samples directory is slightly different on Windows XP and Windows Vista/Windows 7 machines:

Windows XP path: C:\Documents and Settings\<username>\My Documents\ComponentOne Samples\Studio for Silverlight 4.0

Windows Vista and Windows 7 path: C:\Users\<username>\Documents\ComponentOne Samples\Studio for Silverlight 4.0

System Requirements

System requirements for **ComponentOne Studio for Silverlight** include the following:

- Microsoft Silverlight 4.0 or later

- Microsoft Visual Studio 2008 or later

Installing Demonstration Versions

If you wish to try **ComponentOne Studio for Silverlight** and do not have a serial number, follow the steps through the installation wizard and use the default serial number.

The only difference between unregistered (demonstration) and registered (purchased) versions of our products is that the registered version will stamp every application you compile so a ComponentOne banner will not appear when your users run the applications.

Uninstalling Studio for Silverlight

To uninstall **ComponentOne Studio for Silverlight**:

1. Open the **Control Panel** and select **Add or Remove Programs** (XP) or **Programs and Features** (Windows 7/Vista).
2. Select **ComponentOne Studio for Silverlight 4.0** and click the **Remove** button.
3. Click **Yes** to remove the program.

End-User License Agreement

All of the ComponentOne licensing information, including the ComponentOne end-user license agreements, frequently asked licensing questions, and the ComponentOne licensing model, is available online at http://www.componentone.com/SuperPages/Licensing/.

Licensing FAQs

This section describes the main technical aspects of licensing. It may help the user to understand and resolve licensing problems he may experience when using ComponentOne .NET and ASP.NET products.

What is Licensing?

Licensing is a mechanism used to protect intellectual property by ensuring that users are authorized to use software products.

Licensing is not only used to prevent illegal distribution of software products. Many software vendors, including ComponentOne, use licensing to allow potential users to test products before they decide to purchase them.

Without licensing, this type of distribution would not be practical for the vendor or convenient for the user. Vendors would either have to distribute evaluation software with limited functionality, or shift the burden of managing software licenses to customers, who could easily forget that the software being used is an evaluation version and has not been purchased.

How does Licensing Work?

ComponentOne uses a licensing model based on the standard set by Microsoft, which works with all types of components.

Note: The **Compact Framework** components use a slightly different mechanism for run-time licensing than the other ComponentOne components due to platform differences.

When a user decides to purchase a product, he receives an installation program and a Serial Number. During the installation process, the user is prompted for the serial number that is saved on the system. (Users can also enter the serial number by clicking the **License** button on the **About Box** of any ComponentOne product, if available, or by rerunning the installation and entering the serial number in the licensing dialog box.)

When a licensed component is added to a form or Web page, Visual Studio obtains version and licensing information from the newly created component. When queried by Visual Studio, the component looks for licensing information stored in the system and generates a run-time license and version information, which Visual Studio saves in the following two files:

- An assembly resource file which contains the actual run-time license.
- A "licenses.licx" file that contains the licensed component strong name and version information.

These files are automatically added to the project.

In WinForms and ASP.NET 1.x applications, the run-time license is stored as an embedded resource in the assembly hosting the component or control by Visual Studio. In ASP.NET 2.x applications, the run-time license may also be stored as an embedded resource in the **App_Licenses.dll** assembly, which is used to store all run-time licenses for all components directly hosted by WebForms in the application. Thus, the **App_licenses.dll** must always be deployed with the application.

The **licenses.licx** file is a simple text file that contains strong names and version information for each of the licensed components used in the application. Whenever Visual Studio is called upon to rebuild the application resources, this file is read and used as a list of components to query for run-time licenses to be embedded in the appropriate assembly resource. Note that editing or adding an appropriate line to this file can force Visual Studio to add run-time licenses of other controls as well.

Note that the **licenses.licx** file is usually not shown in the Solution Explorer; it appears if you press the **Show All Files** button in the Solution Explorer's Toolbox or, from Visual Studio's main menu, select **Show All Files** on the **Project** menu.

Later, when the component is created at run time, it obtains the run-time license from the appropriate assembly resource that was created at design time and can decide whether to simply accept the run-time license, to throw an exception and fail altogether, or to display some information reminding the user that the software has not been licensed.

All ComponentOne products are designed to display licensing information if the product is not licensed. None will throw licensing exceptions and prevent applications from running.

Common Scenarios

The following topics describe some of the licensing scenarios you may encounter.

Creating components at design time

This is the most common scenario and also the simplest: the user adds one or more controls to the form, the licensing information is stored in the **licenses.licx** file, and the component works.

Note that the mechanism is exactly the same for Windows Forms and Web Forms (ASP.NET) projects.

Creating components at run time

This is also a fairly common scenario. You do not need an instance of the component on the form, but would like to create one or more instances at run time.

In this case, the project will not contain a **licenses.licx** file (or the file will not contain an appropriate run-time license for the component) and therefore licensing will fail.

To fix this problem, add an instance of the component to a form in the project. This will create the **licenses.licx** file and things will then work as expected. (The component can be removed from the form after the **licenses.licx** file has been created).

Adding an instance of the component to a form, then removing that component, is just a simple way of adding a line with the component strong name to the **licenses.licx** file. If desired, you can do this manually using notepad or Visual Studio itself by opening the file and adding the text. When Visual Studio recreates the application resources, the component will be queried and its run-time license added to the appropriate assembly resource.

Inheriting from licensed components

If a component that inherits from a licensed component is created, the licensing information to be stored in the form is still needed. This can be done in two ways:

- Add a **LicenseProvider** attribute to the component.

  This will mark the derived component class as licensed. When the component is added to a form, Visual Studio will create and manage the **licenses.licx** file and the base class will handle the licensing process as usual. No additional work is needed. For example:

```
    [LicenseProvider(typeof(LicenseProvider))]
    class MyGrid: C1.Win.C1FlexGrid.C1FlexGrid
    {
      // ...
    }
```

- Add an instance of the base component to the form.

  This will embed the licensing information into the **licenses.licx** file as in the previous scenario and the base component will find it and use it. As before, the extra instance can be deleted after the **licenses.licx** file has been created.

Please note that ComponentOne licensing will not accept a run-time license for a derived control if the run-time license is embedded in the same assembly as the derived class definition and the assembly is a DLL. This restriction is necessary to prevent a derived control class assembly from being used in other applications without a design-time license. If you create such an assembly, you will need to take one of the actions previously described create a component at run time.

Using licensed components in console applications

When building console applications, there are no forms to add components to and therefore Visual Studio won't create a **licenses.licx** file.

In these cases, create a temporary Windows Forms application and add all the desired licensed components to a form. Then close the Windows Forms application and copy the **licenses.licx** file into the console application project.

Make sure the **licenses.licx** file is configured as an embedded resource. To do this, right-click the **licenses.licx** file in the Solution Explorer window and select **Properties**. In the Properties window, set the **Build Action** property to **Embedded Resource**.

Using licensed components in Visual C++ applications

There is an issue in VC++ 2003 where the **licenses.licx** is ignored during the build process; therefore, the licensing information is not included in VC++ applications.

To fix this problem, extra steps must be taken to compile the licensing resources and link them to the project. Note the following:

1. Build the C++ project as usual. This should create an EXE file and also a licenses.licx file with licensing information in it.
2. Copy the **licenses.licx** file from the application directory to the target folder (**Debug** or **Release**).
3. Copy the **C1Lc.exe** utility and the licensed DLLs to the target folder. (Don't use the standard lc.exe, it has bugs.)
4. Use **C1Lc.exe** to compile the **licenses.licx** file. The command line should look like this:
```
c1lc /target:MyApp.exe /complist:licenses.licx /i:C1.Win.C1FlexGrid.dll
```
5. Link the licenses into the project. To do this, go back to Visual Studio, right-click the project, select **Properties**, and go to the **Linker/Command Line** option. Enter the following:
```
/ASSEMBLYRESOURCE:Debug\MyApp.exe.licenses
```

6. Rebuild the executable to include the licensing information in the application.

Using licensed components with automated testing products

Automated testing products that load assemblies dynamically may cause them to display license dialog boxes. This is the expected behavior since the test application typically does not contain the necessary licensing information and there is no easy way to add it.

This can be avoided by adding the string "C1CheckForDesignLicenseAtRuntime" to the **AssemblyConfiguration** attribute of the assembly that contains or derives from ComponentOne controls. This attribute value directs the ComponentOne controls to use design-time licenses at run time.

For example:

```
#if AUTOMATED_TESTING
   [AssemblyConfiguration("C1CheckForDesignLicenseAtRuntime")]
#endif
   public class MyDerivedControl : C1LicensedControl
   {
         // ...
   }
```

Note that the **AssemblyConfiguration** string may contain additional text before or after the given string, so the **AssemblyConfiguration** attribute can be used for other purposes as well. For example:

```
[AssemblyConfiguration("C1CheckForDesignLicenseAtRuntime,BetaVersion")]
```

THIS METHOD SHOULD ONLY BE USED UNDER THE SCENARIO DESCRIBED. It requires a design-time license to be installed on the testing machine. Distributing or installing the license on other computers is a violation of the EULA.

Troubleshooting

We try very hard to make the licensing mechanism as unobtrusive as possible, but problems may occur for a number of reasons.

Below is a description of the most common problems and their solutions.

I have a licensed version of a ComponentOne product but I still get the splash screen when I run my project.

If this happens, there may be a problem with the **licenses.licx** file in the project. It either doesn't exist, contains wrong information, or is not configured correctly.

First, try a full rebuild (**Rebuild All** from the Visual Studio **Build** menu). This will usually rebuild the correct licensing resources.

If that fails follow these steps:

1. Open the project and go to the Solution Explorer window.
2. Click the **Show All Files** button on the top of the window.
3. Find the **licenses.licx** file and open it. If prompted, continue to open the file.
4. Change the version number of each component to the appropriate value. If the component does not appear in the file, obtain the appropriate data from another **licenses.licx** file or follow the alternate procedure following.
5. Save the file, then close the **licenses.licx** tab.
6. Rebuild the project using the **Rebuild All** option (not just **Rebuild**).

Alternatively, follow these steps:

1. Open the project and go to the Solution Explorer window.

2. Click the **Show All Files** button on the top of the window.
3. Find the **licenses.licx** file and delete it.
4. Close the project and reopen it.
5. Open the main form and add an instance of each licensed control.
6. Check the Solution Explorer window, there should be a **licenses.licx** file there.
7. Rebuild the project using the **Rebuild All** option (not just **Rebuild**).

For ASP.NET 2.x applications, follow these steps:

1. Open the project and go to the Solution Explorer window.
2. Find the **licenses.licx** file and right-click it.
3. Select the **Rebuild Licenses** option (this will rebuild the **App_Licenses.licx** file).
4. Rebuild the project using the **Rebuild All** option (not just **Rebuild**).

I have a licensed version of a ComponentOne product on my Web server but the components still behave as unlicensed.

There is no need to install any licenses on machines used as servers and not used for development.

The components must be licensed on the development machine, therefore the licensing information will be saved into the executable (.exe or .dll) when the project is built. After that, the application can be deployed on any machine, including Web servers.

For ASP.NET 2.x applications, be sure that the App_Licenses.dll assembly created during development of the application is deployed to the bin application bin directory on the Web server.

If your ASP.NET application uses WinForms user controls with constituent licensed controls, the runtime license is embedded in the WinForms user control assembly. In this case, you must be sure to rebuild and update the user control whenever the licensed embedded controls are updated.

I downloaded a new build of a component that I have purchased, and now I'm getting the splash screen when I build my projects.

Make sure that the serial number is still valid. If you licensed the component over a year ago, your subscription may have expired. In this case, you have two options:

Option 1 – Renew your subscription to get a new serial number.

If you choose this option, you will receive a new serial number that you can use to license the new components (from the installation utility or directly from the **About Box**).

The new subscription will entitle you to a full year of upgrades and to download the latest maintenance builds directly from http://prerelease.componentone.com/.

Option 2 – Continue to use the components you have.

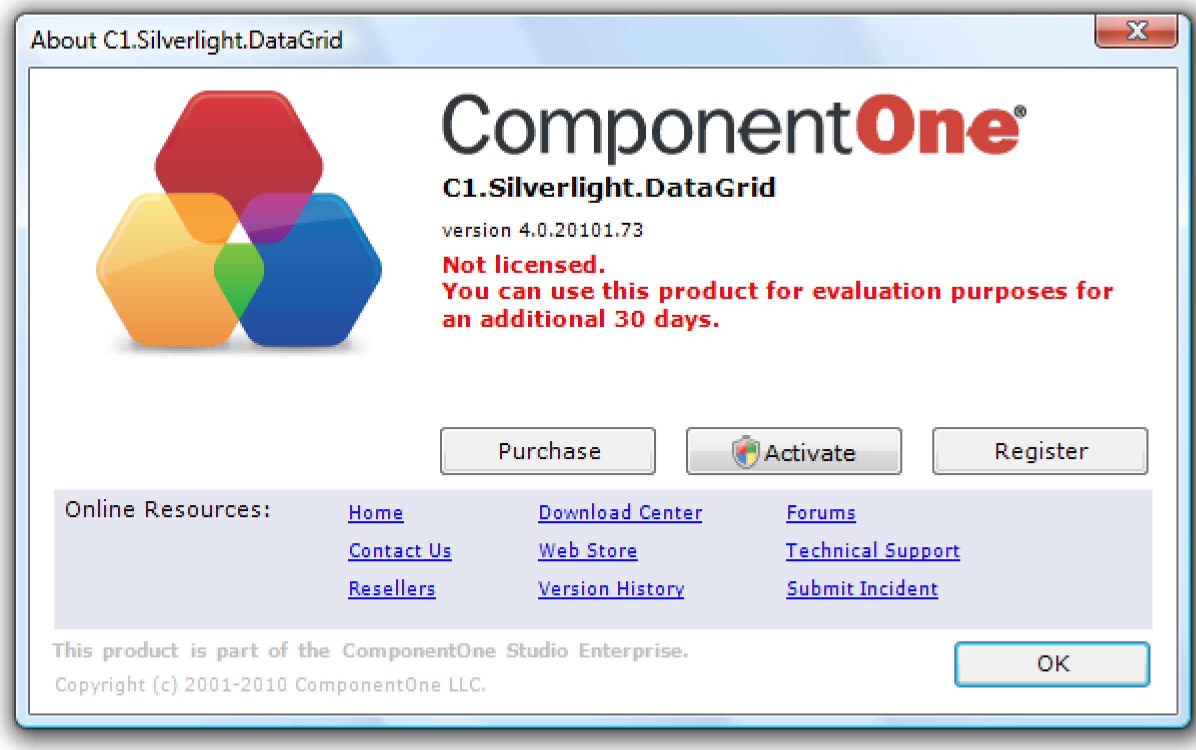
Subscriptions expire, products do not. You can continue to use the components you received or downloaded while your subscription was valid.

Studio for Silverlight Licensing

Licensing for **ComponentOne Studio for Silverlight** is similar to licensing in other ComponentOne products but there are a few differences to note.

Initially licensing in handled similarly to other ComponentOne products. When a user decides to purchase a product, he receives an installation program and a Serial Number. During the installation process, the user is prompted for the serial number that is saved on the system.

In **ComponentOne Studio for Silverlight**, when a control is dropped on a form, a license nag dialog box appears one time. The nag screen appears similar to the following image:


About C1.Silverlight.DataGrid

ComponentOne

C1.Silverlight.DataGrid

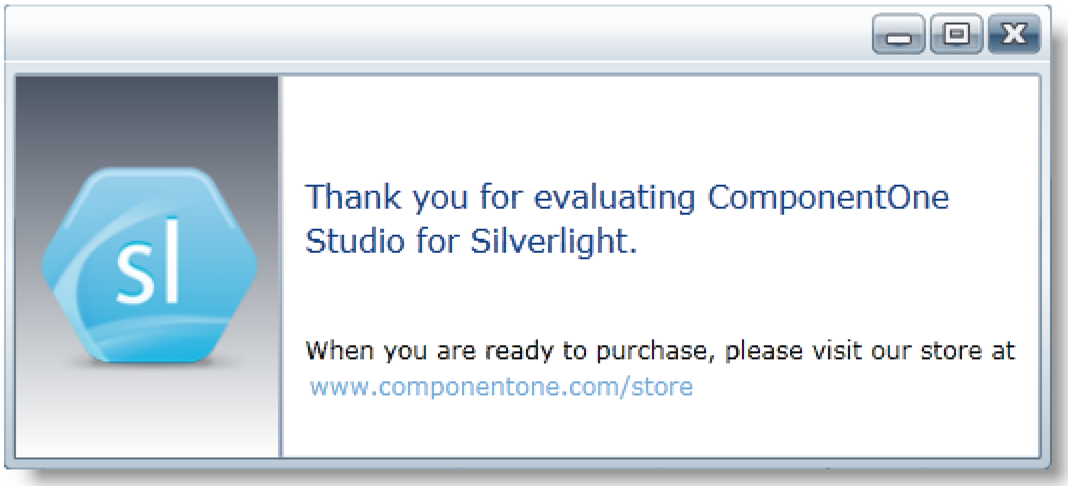
version 4.0.20101.73

Not licensed.
You can use this product for evaluation purposes for an additional 30 days.

This product is part of the ComponentOne Studio Enterprise.
Copyright (c) 2001-2010 ComponentOne LLC.

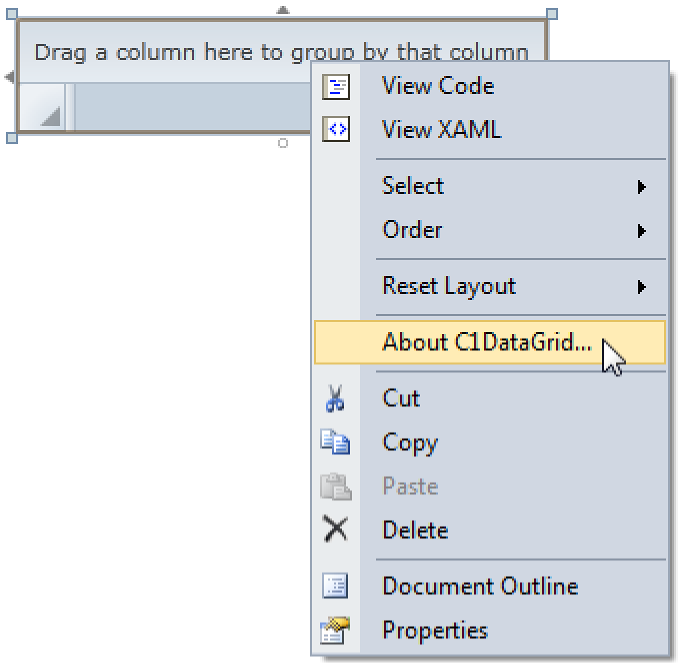
The **About** dialog box displays version information, online resources, and (if the control is unlicensed) buttons to purchase, activate, and register the product.

All ComponentOne products are designed to display licensing information at run time if the product is not licensed. None will throw licensing exceptions and prevent applications from running. Each time an unlicensed Silverlight application is run; end-users will see the following pop-up dialog box:


Thank you for evaluating ComponentOne Studio for Silverlight.

When you are ready to purchase, please visit our store at www.componentone.com/store

To stop this message from appearing, enter the product's serial number by clicking the **Activate** button on the **About** dialog box of any ComponentOne product, if available, or by rerunning the installation and entering the serial number in the licensing dialog box. To open the **About** dialog box, right-click the control and select the **About** option:



Note that when the user modifies any property of a ComponentOne Silverlight control in Visual Studio or Blend, the product will check if a valid license is present. If the product is not currently licensed, an attached property will be added to the control (the **C1NagScreen.Nag** property). Then, when the application executed, the product will check if that property is set, and show a nag screen if the **C1NagScreen.Nag** property is set to **True**. If the user has a valid license the property is not added or is just removed.

One important aspect of this of this process is that the user should manually remove all instances of **c1:C1NagScreen.Nag="true"** in the XAML markup in all files after registering the license (or re-open all the files that include ComponentOne controls in any of the editors). This will ensure that the nag screen does not appear when the application is run.

Technical Support

ComponentOne offers various support options. For a complete list and a description of each, visit the ComponentOne Web site at http://www.componentone.com/SuperProducts/SupportServices/.

Some methods for obtaining technical support include:

- **Online Resources**
  ComponentOne provides customers with a comprehensive set of technical resources in the form of FAQs, samples and videos, Version Release History, searchable Knowledge base, searchable Online Help and more. We recommend this as the first place to look for answers to your technical questions.
- **Online Support via our Incident Submission Form**
  This online support service provides you with direct access to our Technical Support staff via an online incident submission form. When you submit an incident, you'll immediately receive a response via e-mail confirming

that you've successfully created an incident. This email will provide you with an Issue Reference ID and will provide you with a set of possible answers to your question from our Knowledgebase. You will receive a response from one of the ComponentOne staff members via e-mail in 2 business days or less.

- **Product Forums**
  ComponentOne's product forums are available for users to share information, tips, and techniques regarding ComponentOne products. ComponentOne developers will be available on the forums to share insider tips and technique and answer users' questions. Please note that a ComponentOne User Account is required to participate in the ComponentOne Product Forums.

- **Installation Issues**
  Registered users can obtain help with problems installing ComponentOne products. Contact technical support by using the online incident submission form or by phone (412.681.4738). Please note that this does not include issues related to distributing a product to end-users in an application.

- **Documentation**
  Microsoft integrated ComponentOne documentation can be installed with each of our products, and documentation is also available online. If you have suggestions on how we can improve our documentation, please email the Documentation team. Please note that e-mail sent to the Documentation team is for documentation feedback only. Technical Support and Sales issues should be sent directly to their respective departments.

Note: You must create a ComponentOne Account and register your product with a valid serial number to obtain support using some of the above methods.

Redistributable Files

ComponentOne DataGrid for Silverlight is developed and published by ComponentOne LLC. You may use it to develop applications in conjunction with Microsoft Visual Studio or any other programming environment that enables the user to use and integrate the control(s). You may also distribute, free of royalties, the following Redistributable Files with any such application you develop to the extent that they are used separately on a single CPU on the client/workstation side of the network:

- C1.Silverlight.dll
- C1.Silverlight.DataGrid.dll
- C1.Silverlight.DataGrid.Filters.dll
- C1.Silverlight.DataGrid.Summaries.dll

Site licenses are available for groups of multiple developers. Please contact Sales@ComponentOne.com for details.

About this Documentation

You can create your applications using Microsoft Expression Blend or Visual Studio, but Blend is currently the only design-time environment that allows users to design XAML documents visually. In this documentation, we will use the **Design** workspace of Blend for most examples.

Acknowledgements

Microsoft, Windows, Windows Vista, Microsoft Expression, Visual Studio, and Silverlight, are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries. Firefox is a registered trademark of the Mozilla Foundation. Safari is a trademark of Apple Inc., registered in the U.S. and other countries.

ComponentOne

If you have any suggestions or ideas for new features or controls, please call us or write:

Corporate Headquarters

ComponentOne LLC

201 South Highland Avenue

3rd Floor

Pittsburgh, PA 15206 • USA

412.681.4343

412.681.4384 (Fax)

http://www.componentone.com/

ComponentOne Doc-To-Help

This documentation was produced using ComponentOne Doc-To-Help® Enterprise.

XAML and XAML Namespaces

XAML is a declarative XML-based language that is used as a user interface markup language in Silverlight and Windows Presentation Foundation (WPF) and the .NET Framework 3.0 or later. With XAML you can create a graphically rich customized user interface, perform data binding, and much more. For more information on XAML, please see http://www.microsoft.com.

XAML Namespaces

Namespaces organize the objects defined in an assembly. Assemblies can contain multiple namespaces, which can in turn contain other namespaces. Namespaces prevent ambiguity and simplify references when using large groups of objects such as class libraries.

When you create a Microsoft Expression Blend project, a XAML file is created for you and some initial namespaces are specified:

Namespace	Description
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"	This is the default Windows Presentation Foundation namespace.
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"	This is a XAML namespace that is mapped to the **x:** prefix. The **x:** prefix provides a quick, easy way to reference the namespace, which defines many commonly-used features necessary for WPF applications.

When you add a C1DataGrid control to the window in Microsoft Expression Blend or Visual Studio, **Blend** or **Visual Studio** automatically creates an XML namespace for the control. The namespace looks like the following in Microsoft Expression Blend:

```
xmlns:c1="http://schemas.componentone.com/winfx/2006/xaml"
```

The namespace value is **c1**. This is a unified namespace; once this is in the project, all ComponentOne WPF and Silverlight controls found in your references will be accessible through XAML (and IntelliSense). Note that you still need to add references to the assemblies for each control you need to use.

You can also choose to create your own custom name for the namespace. For example:

```
xmlns:MyGrid="http://schemas.componentone.com/winfx/2006/xaml"
```

You can now use your custom namespace when assigning properties, methods, and events. For example, use the following XAML to add a border around the grid:

```
<MyGrid:C1DataGrid Name="C1DataGrid1" BorderThickness="10,10,10,10">
```

Introduction to Silverlight

The following topics detail information about getting started with Silverlight, including Silverlight resources, and general information about templates and deploying Silverlight files.

Silverlight Resources

This help file focuses on **ComponentOne Studio for Silverlight**. For general help on getting started with Silverlight, we recommend the following resources:

- http://www.silverlight.net

  The official Silverlight site, with many links to downloads, samples, tutorials, and more.

- http://silverlight.net/learn/tutorials.aspx

  Silverlight tutorials by Jesse Liberty. Topics covered include:

  - Tutorial 1: Silverlight User Interface Controls
  - Tutorial 2: Data Binding
  - Tutorial 3: Displaying SQL Database Data in a DataGrid using LINQ and WCF
  - Tutorial 4: User Controls
  - Tutorial 5: Styles, Templates and Visual State Manager
  - Tutorial 6: Expression Blend for Developers
  - Tutorial 7: DataBinding & DataTemplates Using Expression Blend
  - Tutorial 8: Multi-page Applications
  - Tutorial 9: ADO.NET DataEntities and WCF Feeding a Silverlight DataGrid
  - Tutorial 10: Hyper-Video

- http://timheuer.com/blog/articles/getting-started-with-silverlight-development.aspx

  Silverlight tutorials by Tim Heuer. Topics covered include:

  - Part 1: Really getting started – the tools you need and getting your first Hello World
  - Part 2: Defining UI Layout – understanding layout and using Blend to help
  - Part 3: Accessing data – how to get data from where
  - Part 4: Binding the data – once you get the data, how can you use it?
  - Part 5: Integrating additional controls – using controls that aren't a part of the core
  - Part 6: Polishing the UI with styles and templates
  - Part 7: Taking the application out-of-browser

- http://weblogs.asp.net/scottgu/pages/silverlight-posts.aspx

  Scott Guthrie's Silverlight Tips, Tricks, Tutorials and Links Page. A useful resource, this page links to several tutorials and samples.

- http://weblogs.asp.net/scottgu/archive/2008/02/22/first-look-at-silverlight-2.aspx

  An excellent eight-part tutorial by Scott Guthrie, covering the following topics:

  - Part 1: Creating "Hello World" with Silverlight 2 and VS 2008

    - Part 2: Using Layout Management
    - Part 3: Using Networking to Retrieve Data and Populate a DataGrid
    - Part 4: Using Style Elements to Better Encapsulate Look and Feel
    - Part 5: Using the ListBox and DataBinding to Display List Data
    - Part 6: Using User Controls to Implement Master/Details Scenarios
    - Part 7: Using Templates to Customize Control Look and Feel
    - Part 8: Creating a Digg Desktop Version of our Application using WPF
- http://blogs.msdn.com/corrinab/archive/2008/03/11/silverlight-2-control-skins.aspx

  A practical discussion of skinning Silverlight controls and applications by Corrina Barber.

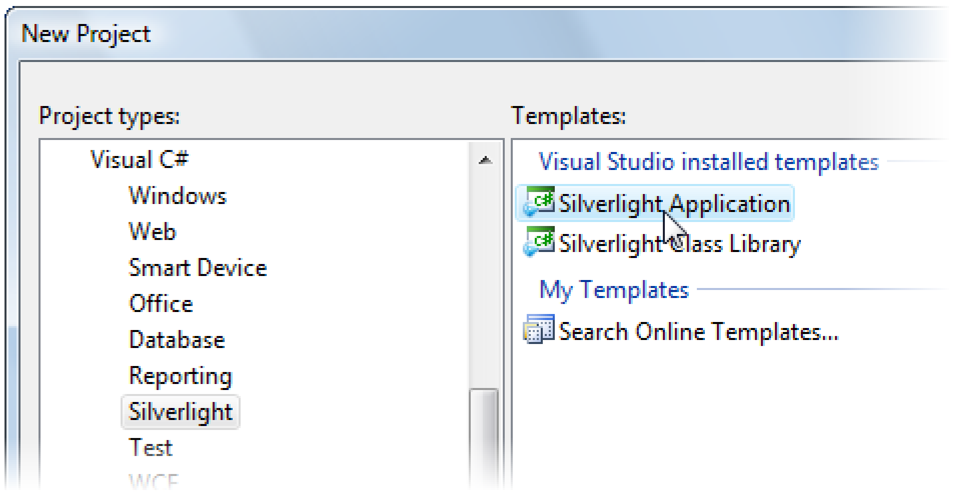
Creating a New Silverlight Project

The following topic details how to create a new Silverlight project in Microsoft Visual Studio 2008 and in Microsoft Expression Blend 3.

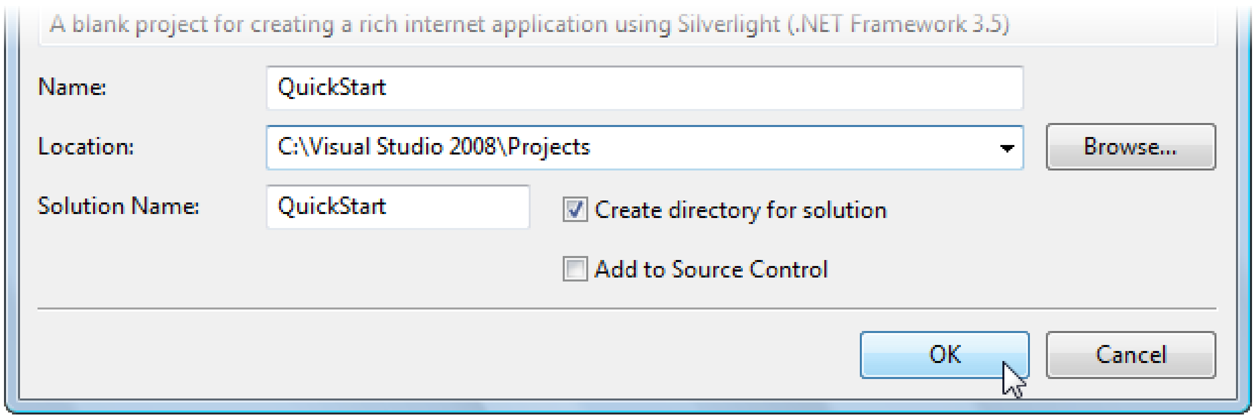
In Visual Studio 2008

Complete the following steps to create a new Silverlight project in Microsoft Visual Studio 2008:

1. Select **File | New | Project** to open the **New Project** dialog box in Visual Studio 2008.
2. In the **Project types** pane, expand either the **Visual Basic** or **Visual C#** node and select **Silverlight**.
3. Choose **Silverlight Application** in the **Templates** pane.



4. Name the project, specify a location for the project, and click **OK**.



A blank project for creating a rich internet application using Silverlight (.NET Framework 3.5)

Name: QuickStart

Location: C:\Visual Studio 2008\Projects

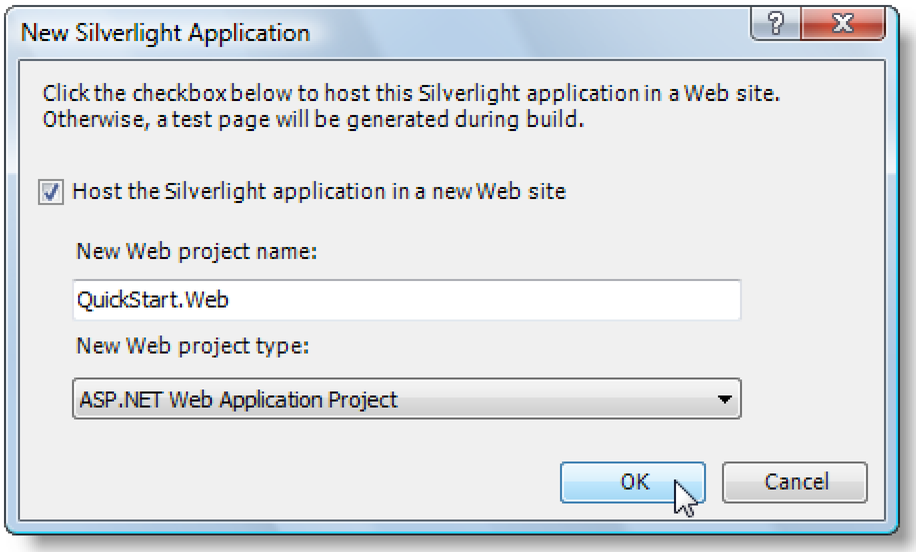
Solution Name: QuickStart

[✓] Create directory for solution

[ ] Add to Source Control

Next, Visual Studio will prompt you for the type of hosting you want to use for the new project.

5. In the **NewSilverlight Application** dialog box, select **OK** to accept the default name and options and to create the project.



New Silverlight Application

Click the checkbox below to host this Silverlight application in a Web site. Otherwise, a test page will be generated during build.

[✓] Host the Silverlight application in a new Web site

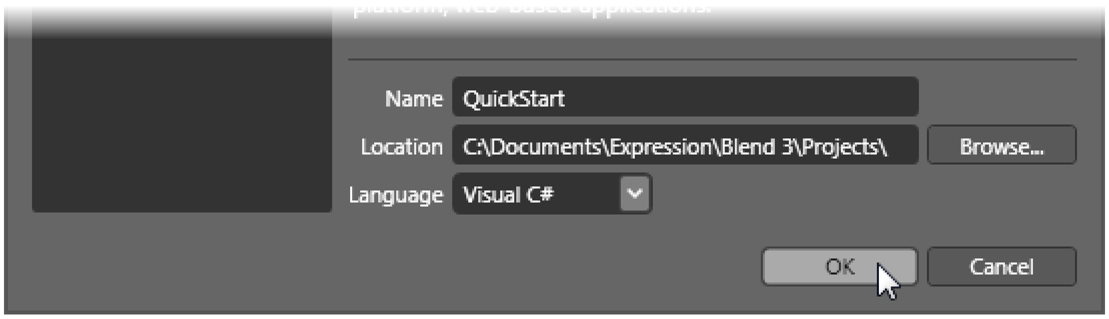
New Web project name: QuickStart.Web

New Web project type: ASP.NET Web Application Project

In Expression Blend 4

Complete the following steps to create a new Silverlight project in Microsoft Expression Blend 4:

6. Select **File | New Project** to open the **New Project** dialog box in Blend 4.
7. In the **Project types** pane, click the **Silverlight** node.
8. In the right pane, choose **Silverlight Application + Website** in the **Templates** pane to create a project with an associated Web site.
9. Name the project, specify a location for the project, choose a language (**Visual C#** or **Visual Basic**), and click **OK**.



Name: QuickStart
Location: C:\Documents\Expression\Blend 3\Projects\
Language: Visual C#

Your new project will be created.

The Project

The solution you just created will contain two projects, **YourProject** and **YourProject.Web**:

- **YourProject**: This is the Silverlight application proper. It will produce a XAP file that gets downloaded to the client and runs inside the Silverlight plug-in.
- **YourProject.Web**: This is the host application. It runs on the server and provides support for the Silverlight application.

Using Templates

The previous sections focused on the **ComponentOne Studio for Silverlight** controls. The following topics focus on Data and Control Templates, and how they are applied to Silverlight controls in general (including controls provided by Microsoft). If you are an experienced Silverlight developer, this information may be of no interest to you.

Data Templates

DataTemplates are a powerful feature in Silverlight. They are virtually identical to the **DataTemplates** in WPF, so if you know WPF there's nothing really new about them.

On the other hand, if you have never used WPF and have seen pieces of XAML that contain styles and templates, you may be confused by the concepts and notation. The good news is DataTemplates are very powerful and are not overly complicated. Once you start using them, the concept will make sense in a couple of minutes and you will be on your way. Remember, just reading the tutorial probably won't be enough to fully grasp the concept. After reading, you should play with the projects.

Create the "Templates" Solution

To illustrate the power of DataTemplates, let's create a new Silverlight solution. Call it "Templates". Complete the following steps:

1. Select **File | New | Project** to open the **New Project** dialog box in Visual Studio 2008.
2. In the **Project types** pane, expand either the **Visual Basic** or **Visual C#** node and select **Silverlight**.
3. Choose **Silverlight Application** in the **Templates** pane.
4. Name the project "Templates", specify a location for the project, and click **OK**.

   Next, Visual Studio will prompt you for the type of hosting you want to use for the new project.
5. In the **New Silverlight Application** dialog box, select **OK** to accept the default name ("Templates.Web") and settings and create the project.
6. Right-click the **Templates** project in the Solution Explorer and select **Add Reference**.

7. In the **Add Reference** dialog box locate and select the C1.Silverlight.dll assembly and click **OK** to add a reference to your project.

   This is required since we will be adding C1.Silverlight controls to the page.

8. Now, open the **MainPage.xaml** file in the **Templates** project and paste in the XAML below:

```
<UserControl x:Class="Templates.MainPage"
    xmlns="http://schemas.microsoft.com/client/2007"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:C1_Silverlight="clr-
namespace:C1.Silverlight;assembly=C1.Silverlight">

  <Grid x:Name="LayoutRoot" >
    <Grid.Background>
      <LinearGradientBrush EndPoint="0.5,1" StartPoint="0.5,0">
        <GradientStop Color="#FF7EB9F0"/>
        <GradientStop Color="#FF284259" Offset="1"/>
      </LinearGradientBrush>
    </Grid.Background>

    <!-- Grid layout -->
    <Grid.RowDefinitions>
      <RowDefinition Height="30" />
      <RowDefinition Height="*" />
    </Grid.RowDefinitions>
    <Grid.ColumnDefinitions>
      <ColumnDefinition Width="*"/>
      <ColumnDefinition Width="*"/>
    </Grid.ColumnDefinitions>

    <!-- Page title -->
    <TextBlock Text="Silverlight Templates" Grid.Column="0"
Grid.ColumnSpan="2"
        TextAlignment="Center" FontSize="18" FontWeight="Bold" />

    <!-- ListBox on the left -->
    <StackPanel Grid.Row="1" Margin="5" >
      <TextBlock Text="ListBox Control" />
      <ListBox x:Name="_listBox" />
    </StackPanel>

    <!-- C1ComboBoxes on the right -->
    <StackPanel Grid.Column="2" Grid.Row="1" Margin="5" >
      <TextBlock Text="C1ComboBox Controls" />
      <C1_Silverlight:C1ComboBox x:Name="_cmb1" Margin="0,0,0,5" />
      <C1_Silverlight:C1ComboBox x:Name="_cmb2" Margin="0,0,0,5" />
    </StackPanel>

  </Grid>
</UserControl>
```

   This creates a page with two columns. The left column has a standard **ListBox** control and the right has two **C1ComboBox**es. These are the controls we will populate and style in the following steps.

Populate the Controls

Before we start using templates and styles, let us populate the controls first. To do that, complete the following:

9. Open the **MainPage.xaml.cs** file and paste the following code into the page constructor:

```
public Page()
{
    InitializeComponent();

    // Get list of items
    IEnumerable list = GetItems();

    // Add items to ListBox and in C1ComboBox
    _listBox.ItemsSource = list;
    _cmb1.ItemsSource = list;

    // Show fonts in the other C1ComboBox
    FontFamily[] ff = new FontFamily[]
    {
       new FontFamily("Default font"),
       new FontFamily("Arial"),
       new FontFamily("Courier New"),
       new FontFamily("Times New Roman"),
       new FontFamily("Trebuchet MS"),
       new FontFamily("Verdana")
    };
    _cmb2.ItemsSource = ff;
}
```

The code populates the **ListBox** and both **C1ComboBoxes** by setting their **ItemsSource** property. **ItemsSource** is a standard property present in most controls that support lists of items (**ListBox**, **DataGrid**, **C1ComboBox**, and so on).

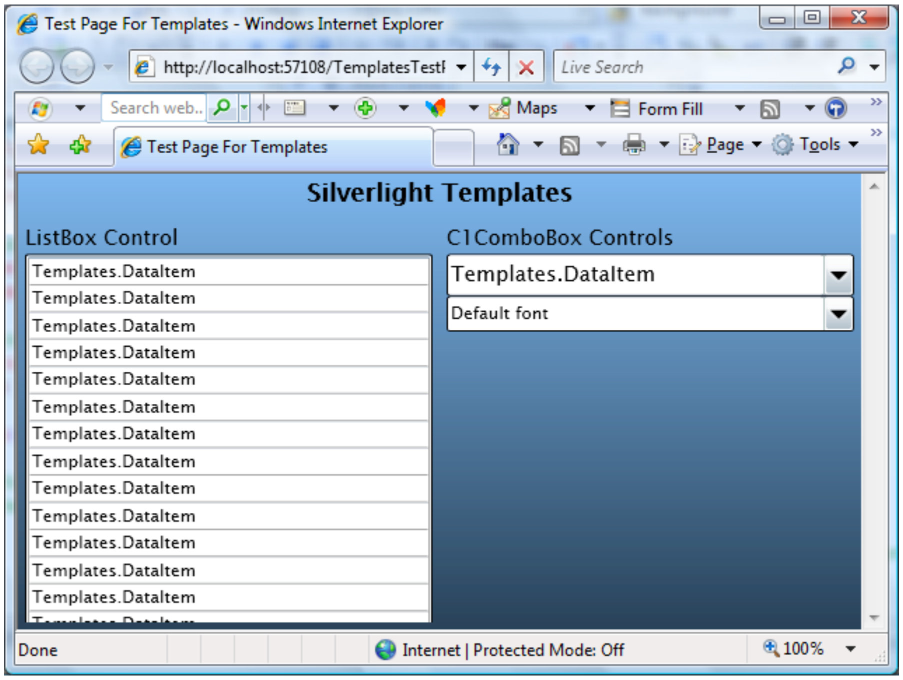
10. Add the following code to implement the **GetItems()** method in the MainPage class:

```
List<DataItem> GetItems()
{
  List<DataItem> members = new List<DataItem>();
  foreach (MemberInfo mi in this.GetType().GetMembers())
  {
    members.Add(new DataItem(mi));
  }
  return members;
}
```

11. Add the definition of the **DataItem** class. to the **MainPage.xaml.cs** file, below the MainPage class definition:

```
public class DataItem
{
    public string ItemName { get; set; }
    public MemberTypes ItemType { get; set; }
    public DataItem(MemberInfo mi)
    {
        ItemName = mi.Name;
        ItemType = mi.MemberType;
    }
}
```

If you run the project now, you will see that the controls are being populated. However, they don't do a very good job of showing the items:


Test Page For Templates - Windows Internet Explorer
http://localhost:57108/TemplatesTest
Silverlight Templates
ListBox Control
C1ComboBox Controls
Templates.DataItem
Default font
Done
Internet | Protected Mode: Off
100%

The controls simply convert the **DataItem** objects into strings using their **ToString()** method, which we didn't override and by default returns a string representation of the object type ("Templates.DataItem").

The bottom **C1ComboBox** displays the font family names correctly. That's because the **FontFamily** class implements the **ToString()** method and returns the font family name.

It is easy to provide a **ToString()** implementation that would return a more useful string, containing one or more properties. For example:

```
public override string ToString()
{
    return string.Format("{0} {1}", ItemType, ItemName);
}
```

If you add this method to the **DataItem** class and run the project again, you will see a slightly more satisfying result. But there's only so much you can do with plain strings. To represent complex objects effectively, we need something more. Enter Data Templates!

Defining and Using Data Templates

Data Templates are objects that map regular .NET objects into **UIElement** objects. They are used by controls that contain lists of regular .NET objects to convert these objects into **UIElement** objects that can be displayed to the user.

For example, the Data Template below can be used to map our **DataItem** objects into a **StackPanel** with two **TextBlock** elements that display the **ItemName** and **ItemType** properties of the **DataItem**. This is what the template definition looks like in XAML markup:

```
<UserControl x:Class="Templates.MainPage"
  xmlns="http://schemas.microsoft.com/client/2007"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  xmlns:C1_Silverlight="clr-
namespace:C1.Silverlight;assembly=C1.Silverlight">

  <!-- Data template used to convert DataItem objects into UIElement
objects -->
```

```
  <UserControl.Resources>
    <DataTemplate x:Key="DataItemTemplate" >
      <StackPanel Orientation="Horizontal" Height="30" >
        <TextBlock Text="{Binding ItemType}"
          Margin="5" VerticalAlignment="Bottom" Foreground="Red"
FontSize="10" />
        <TextBlock Text="{Binding ItemName}"
          Margin="5" VerticalAlignment="Bottom" />
      </StackPanel>
    </DataTemplate>
  </UserControl.Resources>

  <!-- Page content (same as before)... -->
```

This template tells Silverlight (or WPF) that in order to represent a source data object, it should do this:

12. Create a **StackPanel** with two **TextBlock**s in it,
13. Bind the **Text** property of the first **TextBlock** to the **ItemType** property of the source data object, and
14. Bind the **Text** property of the second **TextBlock** object to the **ItemName** property of the source object.

That's it. The template does not specify what type of control can use it (any control can, we will use it with the **ListBox** and also with the **C1ComboBox**), and it does not specify the type of object it should expect (any object will do, as long as it has public properties named **ItemType** and **ItemName**).

To use the template, add an **ItemTemplate** attribute to the controls where you want the template to be applied. In our example, we will apply it to the **ListBox** declaration in the **MainPage.xaml** file:

```
<!-- ListBox on the left -->
<StackPanel Grid.Row="1" Margin="5" >
  <TextBlock Text="ListBox Control" />
  <ListBox x:Name="_listBox"
           ItemTemplate="{StaticResource DataItemTemplate}" />
</StackPanel>
```

And also to the top **C1ComboBox**:

```
<!-- C1ComboBox on the right -->
<StackPanel Grid.Column="2" Grid.Row="1" Margin="5" >
  <TextBlock Text="C1ComboBox Controls" />

  <!-- C1ComboBox 1 -->
  <C1_Silverlight:C1ComboBox x:Name="_cmb1" Margin="0,0,0,5"
        ItemTemplate="{StaticResource DataItemTemplate}" />
```

Note that we can now change the appearance of the **DataItem** objects by modifying the template in one place. Any changes will automatically be applied to all objects that use that template, making application maintenance much easier.

Before you run the application again, let's add a template to the second **C1ComboBox** as well. This control contains a list of font families. We can use templates to display each item using the actual font they represent.

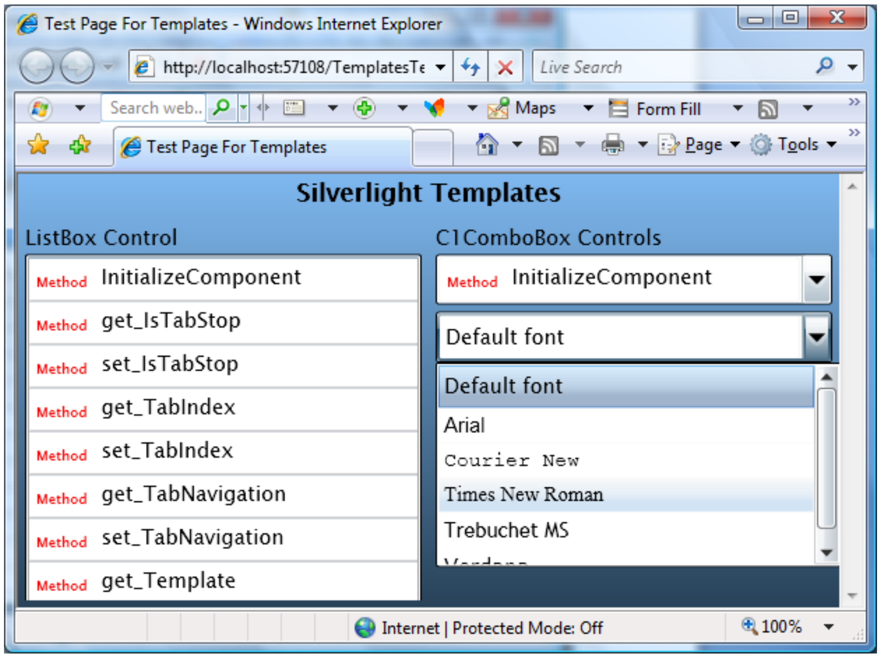
This time, we will not define the template as a resource. It will only be used in one place, so we can insert it inline, as shown below:

```
<!-- C1ComboBox 2 -->
<C1_Silverlight:C1ComboBox x:Name="_cmb2" FontSize="12" Margin="0,0,0,5" >
  <C1_Silverlight:C1ComboBox.ItemTemplate>
    <DataTemplate>
      <TextBlock Text="{Binding}" FontFamily="{Binding}" Margin="4" />
    </DataTemplate>
  </C1_Silverlight:C1ComboBox.ItemTemplate>
</C1_Silverlight:C1ComboBox>
```

Don't let the XAML syntax confuse you. This specifies that in order to create items from data, the control should use a **DataTemplate** that consists of a single **TextBlock** element. The **TextBlock** element should have two of its properties (**Text** and **FontFamily**) bound to the data object itself (as opposed to properties of that object).

In this case, the data object is a **FontFamily** object. Because the template assigns this object to the **Text** property and also to the **FontFamily** property, the **TextBlock** will display the font name and will use the actual font.

If you run the project now, you should see this result:



Note that if you assign a **DataTemplate** to the **C1ComboBox**, it will no longer be able to perform text-related tasks such as auto-search and editing. If you want to re-enable those features, you should provide your own **ItemConverter** that is a standard **TypeConverter**.

Styles and Templates are extremely powerful concepts. We encourage you to play and experiment with this sample. Try modifying the templates to show the data in different ways. The more you experiment, the more comfortable you will feel with these concepts and with the Silverlight/WPF application architecture.

Control Templates

Data Templates allow you to specify how to convert arbitrary data objects into **UIElement** objects that can be displayed to the user. But that's not the only use of templates in Silverlight and WPF. You can also use templates to modify the visual structure of existing **UIElement** objects such as controls.

Most controls have their visual appearance defined by a native XAML resource (typically contained within the assembly that defines the control). This resource specifies a **Style** which assigns values to most of the control's properties, including its **Template** property (which defines the control's internal "visual tree").

For example:

```
<Style TargetType="HyperlinkButton">
  <Setter Property="IsEnabled" Value="true" />
  <Setter Property="IsTabStop" Value="true" />
  <Setter Property="Foreground" Value="#FF417DA5" />
  <Setter Property="Cursor" Value="Hand" />
  <Setter Property="Template">
```

```
    <Setter.Value>
      <ControlTemplate TargetType="HyperlinkButton">
        <Grid x:Name="RootElement" Cursor="{TemplateBinding Cursor}">
          <!-- Focus indicator -->
          <Rectangle x:Name="FocusVisualElement" StrokeDashCap="Round"
...=""/>
          <!-- HyperlinkButton content -->
          <ContentPresenter x:Name="Normal"
               Background="{TemplateBinding Background}"
               Content="{TemplateBinding Content}"
               ContentTemplate="{TemplateBinding ContentTemplate}"...=""/>
        </Grid>
      </ControlTemplate>
    </Setter.Value>
  </Setter>
</Style>
```

This is a very simplified version of the XAML resource used to specify the **HyperlinkButton** control. It consists of a **Style** that begins by setting the default value of several simple properties, and then assigns a value of type **ControlTemplate** to the control's **Template** property.

The **ControlTemplate** in this case consists of a **Grid** (*RootElement*) that contains a **Rectangle** (*FocusVisualElement*) used to indicate the focused state and a **ContentPresenter** (*Normal*) that represents the content portion of the control (and itself contains another **ContentTemplate** property).

Note the *TemplateBinding* attributes in the XAML. These constructs are used to map properties exposed by the control to properties of the template elements. For example, the **Background** property of the hyperlink control is mapped to the **Background** property of the *Normal* element specified in the template.

Specifying controls this way has some advantages. The complete visual appearance is defined in XAML and can be modified by a professional designer using Expression Blend, without touching the code behind it. In practice, this is not as easy as it sounds, because there are logical relationships between the template and the control implementation.

Recognizing this problem, Silverlight introduced a **TemplatePart** attribute that allows control classes to specify the names and types it expects its templates to contain. In the future, this attribute will be added to WPF as well, and used by designer applications such as Blend to validate templates and ensure they are valid for the target control.

For example, the Microsoft **Button** control contains the following **TemplatePart** attributes:

```
/// <summary>
/// Represents a button control, which reacts to the Click event.
/// </summary>
[TemplatePart(Name = Button.ElementRootName, Type =
typeof(FrameworkElement))]
[TemplatePart(Name = Button.ElementFocusVisualName, Type =
typeof(UIElement))]
[TemplatePart(Name = Button.StateNormalName, Type = typeof(Storyboard))]
[TemplatePart(Name = Button.StateMouseOverName, Type =
typeof(Storyboard))]
[TemplatePart(Name = Button.StatePressedName, Type = typeof(Storyboard))]
[TemplatePart(Name = Button.StateDisabledName, Type = typeof(Storyboard))]
public partial class Button : ButtonBase
```

These six template parts constitute a contract between the control implementation and the design specification. They tell the designer that the control implementation expects to find certain elements in the template (defined by their name and type).

Well-behaved controls should degrade gracefully, not crashing if some non-essential elements are missing from the template. For example, if the control can't find a **Storyboard** named *Button.StateMouseOverName* in the template, it should not do anything when the mouse hovers over it.

Well-implemented templates should fulfill the contract and provide all the elements that the control logic supports. Designer applications such as Blend can enforce the contract and warn designers if they try to apply invalid templates to controls.

For the time being, the easiest way to create new templates for existing controls is to start with the original XAML and customize it.

We will not show any actual examples of how to create and use custom control templates here. Instead, we suggest you download the examples developed by Corrina Barber:

http://blogs.msdn.com/corrinab/archive/2008/03/11/silverlight-2-control-skins.aspx

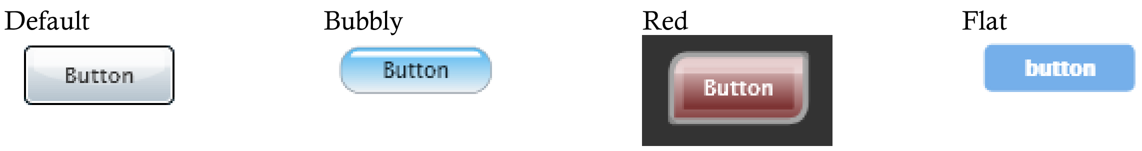
The link contains previews and downloads for three 'skins' (bubbly, red, and flat). Each skin consists of a set of **Style** specifications, similar to the one shown above, which are added to the application's global XAML file (App.xaml). The format is similar to this:

```
<Application xmlns="http://schemas.microsoft.com/client/2007"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    x:Class="Styles_Red.App"

<Application.Resources>
  <!-- Button -->
  <Style x:Key="buttonStyle" TargetType="Button">
    <Setter Property="IsEnabled" Value="true" />
    <Setter Property="IsTabStop" Value="true" />
    <Setter Property="Foreground" Value="#FF1E2B33" />
    <Setter Property="Cursor" Value="Hand" />
    <Setter Property="TextAlignment" Value="Center" />
    <!-- A lot more XAML follows… -->
```

Once these styles are defined in the **App.xaml** file, they can be assigned to any controls in the application:

```
<Button Content="Button" Style="{StaticResource buttonStyle}"/>
```

If you are curious, this is what the **Button** control looks like after applying each of the skins defined in the reference above:



This mechanism is extremely powerful. You can change what the controls look like and even the parts used internally to build them.

Unlike data templates, however, control templates are not simple to create and modify. Creating or changing a control template requires not only design talent but also some understanding of how the control works.

It is also a labor-intensive proposition. In addition to their normal appearance, most controls have **Storyboards** that are applied to change their appearance when the mouse hovers over them, when they gain focus, get pressed, get disabled, and so on (see the **C1ComboBox** example above).

Furthermore, all controls in an application should appear consistent. You probably wouldn't want to mix bubbly buttons with regular scrollbars on the same page for example. So each 'skin' will contain styles for many controls.

Some controls are designed with custom templates in mind. For example, the **C1ComboBox** has an **ItemsPanel** property of type **ItemsPanelTemplate**. You can use this property to replace the default drop-down **ListBox** element with any other **UIElement** you like.

For examples of using the **ItemsPanel** property, check the **ControlExplorer** sample installed by default with **ComponentOne Studio for Silverlight**.

Preparing Your Enterprise Environment

Several considerations are important to take into account when planning a corporate deployment of your Silverlight applications in an enterprise environment. For information about these considerations and a description of system requirements and deployment methods as well as the techniques to maintain and support Silverlight after deployment, please see the Silverlight Enterprise Deployment Guide provided by the Microsoft Silverlight team.

The guide helps you to plan and carry out a corporate deployment of Silverlight, and covers:

- Planning the deployment
- Testing deployment strategy
- Deploying Silverlight
- Maintaining Silverlight in your environment

The Silverlight Enterprise Deployment Guide is available for download from the Silverlight whitepapers site: http://silverlight.net/learn/whitepapers.aspx.

Theming

One of the main advantages to using Silverlight is the ability to change the style or template of any control. Controls are "lookless" with fully customizable user interfaces and the ability to use built-in and custom themes. Themes allow you to customize the appearance of controls and take advantage of Silverlight's XAML-based styling. The following topics introduce you to styling Silverlight controls with themes.

You can customize WPF and Microsoft Silverlight controls by creating and modifying control templates and styles. This results in a unique and consistent look for your application.

Templates and styles define the pieces that make up a control and the default behavior of the control, respectively. You can create templates and styles by making copies of the original styles and templates for a control. Modifying templates and styles is an easy way to essentially make new controls in Design view of Microsoft Expression Blend, without having to use code. The following topics provide a detailed comparison of styles and templates to help you decide whether you want to modify the style or template of a control, or both. The topics also discuss the built-in themes available in **ComponentOne Studio for Silverlight**.

Available Themes

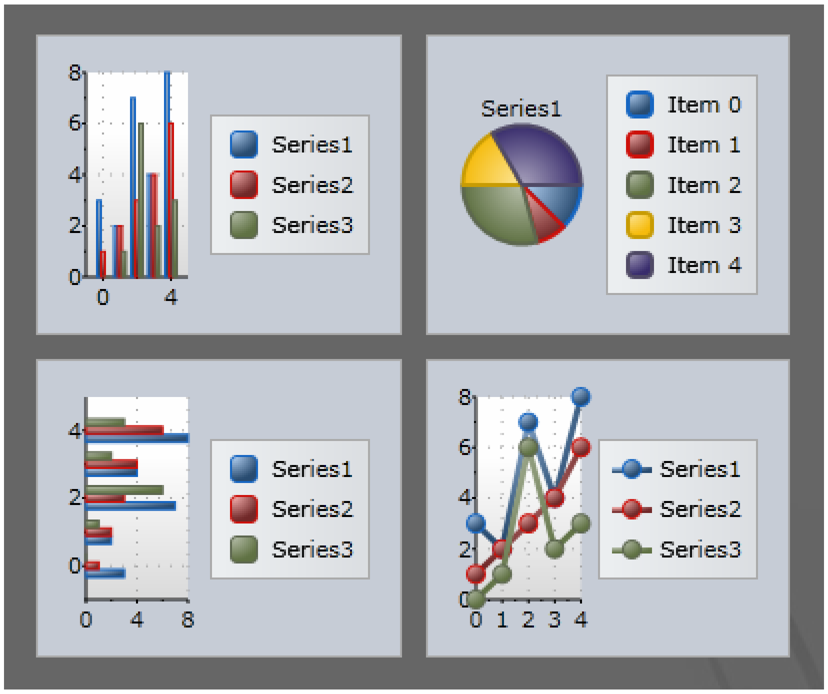
ComponentOne Studio for Silverlight includes several theming options, and several built-in Silverlight Toolkit themes including:

- BureauBlack
- ExpressionDark
- ExpressionLight
- RainierOrange
- ShinyBlue
- WhistlerBlue

Each of these themes is based on themes in the Silverlight Toolkit and installed in its own assembly in the **Studio for Silverlight** installation directory. The following topics detail each built-in theme.

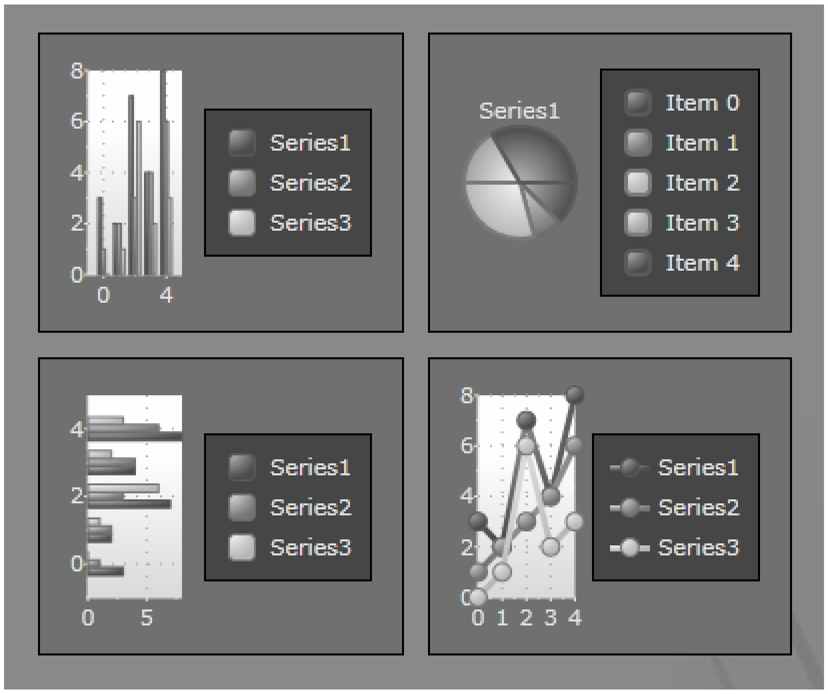
BureauBlack

The BureauBlack theme is a dark colored theme similar to the Microsoft Bureau Black theme included in the Silverlight Toolkit. The BureauBlack theme appears similar the following when applied to the **ComponentOne Studio for Silverlight** charting controls:



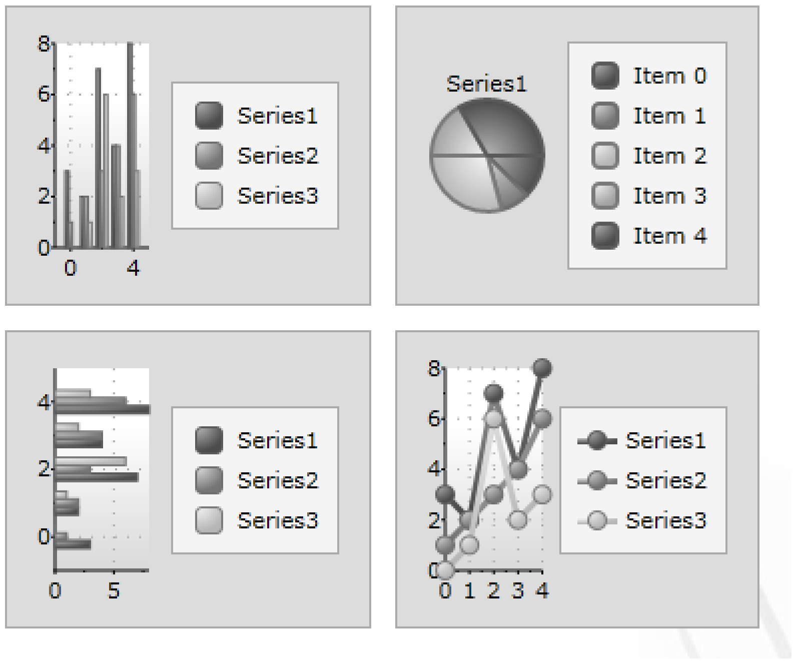
ExpressionDark

The ExpressionDark theme is a grayscale theme based on the Microsoft Expression Dark theme, which is included in the Silverlight Toolkit. For example, the theme appears similar the following when applied to the **ComponentOne Studio for Silverlight** charting controls:



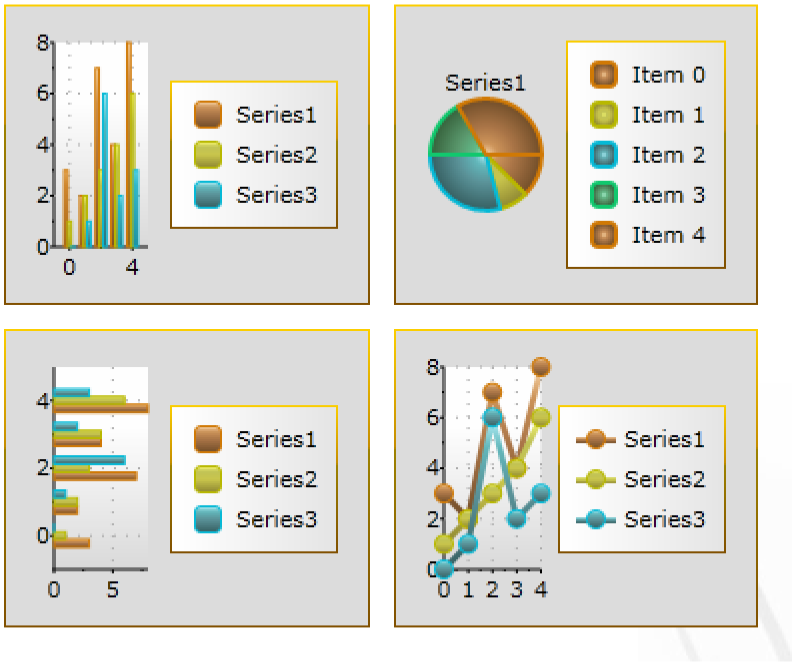
ExpressionLight

The ExpressionLight theme is a grayscale theme based on the Microsoft Expression Light theme, which is included in the Silverlight Toolkit. For example, the theme appears similar the following when applied to the **ComponentOne Studio for Silverlight** charting controls:



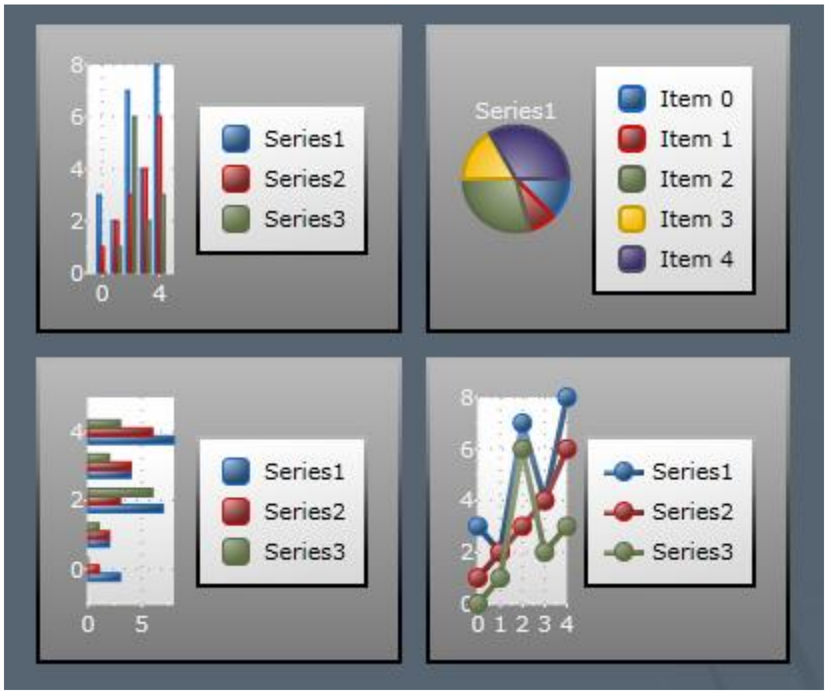
RainierOrange

The RainerOrange theme is an orange-based theme similar to the Microsoft Rainer Orange theme, which is included in the Silverlight Toolkit. The RainerOrange theme appears similar the following when applied to the **ComponentOne Studio for Silverlight** charting controls:



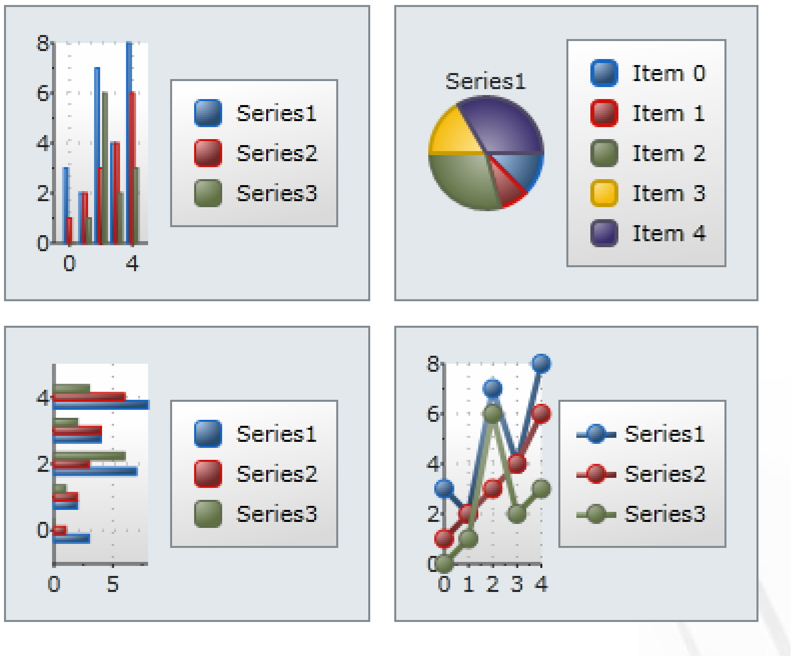
ShinyBlue

The ShinyBlue theme is a blue-based theme similar to the Microsoft Shiny Blue theme included in the Silverlight Toolkit. The ShinyBlue theme appears similar the following when applied to the **ComponentOne Studio for Silverlight** charting controls:



WhistlerBlue

The WhistlerBlue theme is a blue-based theme similar to the Microsoft Whistler Blue theme, which is included in the Silverlight Toolkit. The WhistlerBlue theme appears similar the following when applied to the **ComponentOne Studio for Silverlight** charting controls:



Custom Themes

In addition to using one of the built-in themes, you can create your own custom theme from scratch or create a custom theme based on an existing built-in theme. See Included XAML Files (page 34) for the included files that you can base a theme on.

Included XAML Files

Several auxiliary XAML elements are installed with **ComponentOne Studio for Silverlight**. These elements include templates and themes and are located in the **Studio for Silverlight** installation directory. You can incorporate these elements into your project to, for example, create your own theme based on the included Office 2007 themes.

By default, these files are located in the **generics.zip** file in the **C:\Program Files\ComponentOne\Studio for Silverlight 4.0\Help** folder. Unzip the **generics.zip** file to a folder to see all the XAML files associated with **Studio for Silverlight** controls. In the following topics the included files are listed by assembly with their location folder within the **generics.zip** file noted.

C1.Silverlight.DataGrid

The following XAML file can be used to customize items in the **C1.Silverlight.DataGrid** assembly:

Element	Folder	Description
generic.xaml	C1.Silverlight.DataGrid\themes	Specifies the templates for different styles and the initial style of the controls.

Common.xaml	C1.Silverlight.DataGrid\themes	Specifies attributes for common elements in the controls.
DataGridCellPresenter.xaml	C1.Silverlight.DataGrid\themes	Specifies attributes for common elements in the controls.
DataGridColumnHeaderPresenter.xaml	C1.Silverlight.DataGrid\themes	Specifies attributes for the column header presenter.
DataGridDetailsPresenter.xaml	C1.Silverlight.DataGrid\themes	Specifies attributes for the data details presenter.
DataGridDragNDrop.xaml	C1.Silverlight.DataGrid\themes	Specifies attributes for grid drag-and-drop operation.
DataGridFilter.xaml	C1.Silverlight.DataGrid\themes	Specifies attributes for the grid's filtering.
DataGridGroupingPresenter.xaml	C1.Silverlight.DataGrid\themes	Specifies attributes for the grouping presenter.
DataGridRowHeaderPresenter.xaml	C1.Silverlight.DataGrid\themes	Specifies attributes for the row header presenter.
DataGridRowPresenter.xaml	C1.Silverlight.DataGrid\themes	Specifies attributes for the row presenter.
DataGridVerticalFreezingSeparatorPresenter.xaml	C1.Silverlight.DataGrid\themes	Specifies attributes for the freezing separator presenter.

C1.Silverlight.DataGrid.Ria

The following XAML file can be used to customize items in the **C1.Silverlight.DataGrid.Ria** assembly:

Element	Folder	Description
generic.xaml	C1.Silverlight.DataGrid.Ria\themes	Specifies the templates for different styles and the initial style of the controls.

C1.Silverlight.Theming.BureauBlack

The following XAML files can be used to customize items in the **C1.Silverlight.BureauBlack** assembly:

Element	Folder	Description
BureauBlack.xaml	C1.Silverlight.Theming.BureauBlack	Specifies resources and styling elements for each ComponentOne Silverlight control.
System.Windows.Controls.Theming.BureauBlack.xaml	C1.Silverlight.Theming.BureauBlack	Specifies the standard Microsoft BureauBlack resources and styling elements.
Theme.xaml	C1.Silverlight.Theming.BureauBlack	Specifies the standard resources and styling elements.

C1.Silverlight.Theming.ExpressionDark

The following XAML files can be used to customize items in the **C1.Silverlight.ExpressionDark** assembly:

Element	Folder	Description
ExpressionDark.xaml	C1.Silverlight.Theming.ExpressionDark	Specifies resources and styling elements for each ComponentOne Silverlight control.
System.Windows.Controls.	C1.Silverlight.Themin	Specifies the standard Microsoft ExpressionDark

Theming.ExpressionDark.xaml	g.ExpressionDark	resources and styling elements.
Theme.xaml	C1.Silverlight.Theming.ExpressionDark	Specifies the standard resources and styling elements.

C1.Silverlight.Theming.ExpressionLight

The following XAML files can be used to customize items in the **C1.Silverlight.ExpressionLight** assembly:

Element	Folder	Description
ExpressionLight.xaml	C1.Silverlight.Theming.ExpressionLight	Specifies resources and styling elements for each ComponentOne Silverlight control.
System.Windows.Controls.Theming.ExpressionLight.xaml	C1.Silverlight.Theming.ExpressionLight	Specifies the standard Microsoft ExpressionLight resources and styling elements.
Theme.xaml	C1.Silverlight.Theming.ExpressionLight	Specifies the standard resources and styling elements.

C1.Silverlight.Theming.RainierOrange

The following XAML files can be used to customize items in the **C1.Silverlight.RainierOrange** assembly:

Element	Folder	Description
RainierOrange.xaml	C1.Silverlight.Theming.RainierOrange	Specifies resources and styling elements for each ComponentOne Silverlight control.
Theme.xaml	C1.Silverlight.Theming.RainierOrange	Specifies the standard resources and styling elements.

C1.Silverlight.Theming.ShinyBlue

The following XAML files can be used to customize items in the **C1.Silverlight.ShinyBlue** assembly:

Element	Folder	Description
ShinyBlue.xaml	C1.Silverlight.Theming.ShinyBlue	Specifies resources and styling elements for each ComponentOne Silverlight control.
Theme.xaml	C1.Silverlight.Theming.ShinyBlue	Specifies the standard resources and styling elements.

C1.Silverlight.Theming.WhistlerBlue

The following XAML files can be used to customize items in the **C1.Silverlight.WhistlerBlue** assembly:

Element	Folder	Description
WhistlerBlue.xaml	C1.Silverlight.Theming.WhistlerBlue	Specifies resources and styling elements for each ComponentOne Silverlight control.
Theme.xaml	C1.Silverlight.Theming.WhistlerBlue	Specifies the standard resources and styling elements.

Implicit and Explicit Styles

The following topic detail using implicit and explicit styles and using the **ImplicitStyleManager** which is included in the Silverlight Toolkit. For more information about the Silverlight Toolkit, see CodePlex.

Implicit Styles

If you're familiar with WPF (Windows Presentation Foundation) you may be used to setting styles implicitly so the application has a uniform appearance – for example, you're used to setting the style for all instances of a particular control in the application's resources. Unfortunately Silverlight does not support implicit styles in the same way that WPF does and you would normally have to indicate the style to use in each instance of the control. This can be tedious to do if you have several controls on a page and that's where the **ImplicitStyleManager** comes in handy.

The **ImplicitStyleManager** class is located in the Microsoft.Windows.Controls.Theming namespace (in the Microsoft.Windows.Controls assembly).

WPF and Silverlight Styling

In WPF, you can set styles implicitly. When you set styles implicitly all instances of a particular type can be styled at once. For example, the WPF **C1DropDown** control might be styled with the following markup:

```
<Grid>
    <Grid.Resources>
        <Style TargetType="{x:Type c1:C1DropDown}">
            <Setter Property="Background" Value="Red" />
        </Style>
    </Grid.Resources>
    <c1:C1DropDown Height="30" HorizontalAlignment="Center"
Name="C1DropDown1" VerticalAlignment="Center" Width="100" />
</Grid>
```

This would set the background of the control to be the color red as in the following image:

All **C1DropDown** controls in the grid would also appear red; **C1DropDown** controls outside of the Grid would not appear red. This is what is meant by implicit styles – the style is assigned to all controls of a particular type. Inherited controls would also inherit the style.

Silverlight, however, does not support implicit styles. In Silverlight you could add the style to the Grid's resources similarly:

```
<Grid.Resources>
    <Style x:Key="DropDownStyle" TargetType="c1:C1DropDown">
        <Setter Property="Background" Value="Red" />
    </Style>
</Grid.Resources>
```

But the Silverlight **C1DropDown** control would not be styled unless the style was explicitly set, as in the following example:

```
<c1:C1DropDown Height="30" HorizontalAlignment="Center" Name="C1DropDown1"
VerticalAlignment="Center" Width="100" Style="{StaticResource
DropDownStyle}"/>
```

While this is easy enough to set on one control, if you have several controls it can be tedious to set the style on each one. That's where the **ImplicitStyleManager** comes in. See Using the ImplicitStyleManager (page 38) for more information.

Using the ImplicitStyleManager

The **ImplicitStyleManager** lets you set styles implicitly in Silverlight as you might in WPF. You can find the **ImplicitStyleManger** in the **System.Windows.Controls.Theming.Toolkit.dll** assembly installed with the Silverlight Toolkit.

To use the **ImplicitStyleManager** add a reference in your project to the **System.Windows.Controls.Theming.Toolkit.dll** assembly and add its namespace to the initial **UserControl** tag as in the following markup:

```
<UserControl
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml" xmlns:c1="clr-
namespace:C1.Silverlight;assembly=C1.Silverlight" xmlns:theming="clr-
namespace:System.Windows.Controls.Theming;assembly=System.Windows.Controls.Th
eming.Toolkit" x:Class="C1Theming.MainPage" Width="640" Height="480">
```

Once you've added the reference and namespace you can use the **ImplicitStyleManager** in your application. For example, in the following markup a style is added and implicitly implemented:

```
<Grid x:Name="LayoutRoot" Background="White"
theming:ImplicitStyleManager.ApplyMode="OneTime">

    <Grid.Resources>

        <Style TargetType="c1:C1DropDown">

            <Setter Property="Background" Value="Red" />

        </Style>

    </Grid.Resources>

    <StackPanel HorizontalAlignment="Center" VerticalAlignment="Center">

        <c1:C1DropDown Margin="5" Content="C1DropDown" Height="30"
Width="100"/>

    </StackPanel>

</Grid>
```

Applying Themes to Controls

You can easily customize your application, by applying one of the built-in themes to your ComponentOne Silverlight control. Each of the built-in themes is based on a Silverlight Toolkit theme. For information about each of the built-in themes, see Available Themes (page 30). In this example, you'll add the RainierOrange theme to the **C1DropDown** control on a page.

To apply the theme, complete the following steps:

1. In Visual Studio, select **File | New Project**.
2. In the **New Project** dialog box, select the language in the left pane and in the right-pane select **Silverlight Application**. Enter a **Name** and **Location** for your project and click **OK**.
3. In the **New Silverlight Application** dialog box, leave the default settings and click **OK**.

A new application will be created and should open with the **MainPage.xaml** file displayed in XAML view.

4. Place the mouse cursor between the <`Grid>` and `</Grid>` tags in XAML view.

   You will add the theme and control to the Grid in the next steps.

5. Navigate to the Visual Studio Toolbox and double-click on the **C1ThemeRanierOrange** icon to declare the theme. The theme's namespace will be added to the page and the theme's tags will be added to the Grid in XAML view. The markup will appear similar to the following:

```
<UserControl xmlns:my="clr-
namespace:C1.Silverlight.Theming.RainierOrange;assembly=C1.Silverlight.
Theming.RainierOrange"  x:Class="C1Silverlight.MainPage"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    mc:Ignorable="d" d:DesignWidth="640" d:DesignHeight="480">
  <Grid x:Name="LayoutRoot">
<my:C1ThemeRainierOrange></my:C1ThemeRainierOrange>
  </Grid>
</UserControl>
```

   Any controls that you add within the theme's tags will now be themed.

6. Place your cursor between the `<my:C1ThemeRanierOrange>` and `</my:C1ThemeRanierOrange>` tags.

7. In the Toolbox, double-click the **C1DropDown** icon to add the control to the project. The C1.Silverlight namespace will be added to the page and the control's tags will be added within the theme's tags in XAML view. The markup will appear similar to the following:

```
<UserControl xmlns:c1="clr-
namespace:C1.Silverlight;assembly=C1.Silverlight"
xmlns:my="clr-
namespace:C1.Silverlight.Theming.RainierOrange;assembly=C1.Silverlight.
Theming.RainierOrange"  x:Class="C1Silverlight.MainPage"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    mc:Ignorable="d" d:DesignWidth="640" d:DesignHeight="480">
  <Grid x:Name="LayoutRoot">
<my:C1ThemeRainierOrange>
    <c1:C1DropDown Width="100" Height="30"></c1:C1DropDown>
</my:C1ThemeRainierOrange>
  </Grid>
</UserControl>
```

What You've Accomplished

Run your project and observe that the **C1DropDown** control now appears in the **RainierOrange** theme. Note that you can only set the **Content** property on the theme once, so to theme multiple controls using this method you will need to add a panel, for example a Grid or StackPanel, within the theme and then add multiple controls within the panel.

You can also use the **ImplicitStyleManager** to theme all controls of a particular type. For more information, see Using the ImplicitStyleManager (page 38).

Applying Themes to an Application

The following topic details one method of applying a theme application-wide in Visual Studio. In this topic you'll add a class to your application that initializes a built-in theme. You'll then apply the theme to the MainPage of your application.

To apply the theme, complete the following steps:

1. In Visual Studio, select **File | New Project**.
2. In the **New Project** dialog box, select the language in the left pane and in the right-pane select **Silverlight Application**. Enter a **Name** and **Location** for your project and click **OK**.
3. In the **New Silverlight Application** dialog box, leave the default settings and click **OK**.

   A new application will be created and should open with the **MainPage.xaml** file displayed in XAML view.
4. In the Solution Explorer, right-click the project and choose **Add Reference**.
5. In the **Add Reference** dialog box choose the **C1.Silverlight.Theming** and **C1.Silverlight.Theming.RainierOrange** assemblies and click **OK**.
6. In the Solution Explorer, right-click the project and select **Add | New Item**.
7. In the **Add New Item** dialog box, choose **Class** from the templates list, name the class "MyThemes", and click the **Add** button to create and a new class. The newly created **MyThemes** class will open.
8. Add the following import statements to the top of the class:

   - Visual Basic

     ```
     Imports C1.Silverlight.Theming
     Imports C1.Silverlight.Theming.RainierOrange
     ```

   - C#

     ```
     using C1.Silverlight.Theming;
     using C1.Silverlight.RainierOrange;
     ```

9. Add code to the class so it appears like the following:

   - Visual Basic

     ```
     Public Class MyThemes
         Private _myTheme As C1Theme = Nothing
         Public ReadOnly Property MyTheme() As C1Theme
             Get
                 If _myTheme Is Nothing Then
                     _myTheme = New C1ThemeRainierOrange()
                 End If
                 Return _myTheme
             End Get
         End Property
     End Class
     ```

   - C#

     ```
     public class MyThemes
     {
         private static C1Theme _myTheme = null;
         public static C1Theme MyTheme
         {
             get
             {
                 if (_myTheme == null)
                 _myTheme = new C1ThemeRainierOrange();
     ```

```
                return _myTheme;
            }
        }
}
```

10. In the Solution Explorer, double-click the **App.xaml.vb** or **App.xaml.cs** file.
11. Add the following import statement to the top of the file, where *ProjectName* is the name of your application:
    - Visual Basic
    ```
    Imports ProjectName
    ```
    - C#
    ```
    using ProjectName;
    ```
12. Add code to the **Application_Startup** event of the **App.xaml.vb** or **App.xaml.cs** file so it appears like the following:
    - Visual Basic
    ```
    Private Sub Application_Startup(ByVal o As Object, ByVal e As
    StartupEventArgs) Handles Me.Startup
        Dim MyMainPage As New MainPage()
        Dim themes As New MyThemes
        themes.MyTheme.Apply(MyMainPage)
        Me.RootVisual = MyMainPage
    End Sub
    ```
    - C#
    ```
    private void Application_Startup(object sender, StartupEventArgs e)
    {
        MainPage MyMainPage = new MainPage();
        MyThemes.MyTheme.Apply(MyMainPage);
        this.RootVisual = MyMainPage;
    }
    ```

    Now any control you add to the **MainPage.xaml** file will automatically be themed.
13. Return to the **MainPage.xaml** file and place the mouse cursor between the <`Grid>` and `</Grid>` tags in XAML view.
14. In the Toolbox, double-click the **C1DropDown** icon to add the control to the project.
15. Update the control's markup so it appears like the following:
    ```
    <c1:C1DropDown Width="100" Height="30"></c1:C1DropDown>
    ```

What You've Accomplished

Run your project and observe that the **C1DropDown** control now appears in the **RainierOrange** theme. To change the theme chosen, now all you would need to do is change the theme in the **MyThemes** class.

For example, to change to the ExpressionDark theme:

1. Add a reference to the **C1.Theming.Silverlight.ExpressionDark.dll** assembly.
2. Open the **MyThemes** class in your project and add the following import statements to the top of the class:
    - Visual Basic
    ```
    Imports C1.Silverlight.Theming.ExpressionDark
    ```
    - C#
    ```
    using C1.Silverlight.Theming.ExpressionDark;
    ```
3. Update code in the class so it appears like the following:

- Visual Basic
```
Public Class MyThemes
    Private _myTheme As C1Theme = Nothing
    Public ReadOnly Property MyTheme() As C1Theme
        Get
            If _myTheme Is Nothing Then
                _myTheme = New C1ThemeExpressionDark()
            End If
            Return _myTheme
        End Get
    End Property
End Class
```
- C#
```
public class MyThemes
{
    private static C1Theme _myTheme = null;
    public static C1Theme MyTheme
    {
        get
        {
            if (_myTheme == null)
            _myTheme = new C1ThemeExpressionDark();
            return _myTheme;
        }
    }
}
```

Note that the above steps apply the theme to the **MainPage.xaml** file. To apply the theme to additional pages, you would need to add the following code to each page:

- Visual Basic
```
Dim themes As New MyThemes
themes.MyTheme.Apply(MyMainPage)
```
- C#
```
MyThemes.MyTheme.Apply(LayoutRoot);
```

The theme will then be applied to the page. So, you only have to change one line of code to the class to change the theme, and you only have to add one line of code to each page to apply the theme.

ComponentOne ClearStyle Technology

ComponentOne ClearStyle™ technology is a new, quick and easy approach to providing Silverlight and WPF control styling. ClearStyle allows you to create a custom style for a control without having to deal with the hassle of XAML templates and style resources.

Currently, to add a theme to all standard Silverlight controls, you must create a style resource template. In Microsoft Visual Studio this process can be difficult; this is why Microsoft introduced Expression Blend to make the task a bit easier. Having to jump between two environments can be a bit challenging to developers who are not familiar with Blend or do not have the time to learn it. You could hire a designer, but that can complicate things when your designer and your developers are sharing XAML files.

That's where ClearStyle comes in. With ClearStyle the styling capabilities are brought to you in Visual Studio in the most intuitive manner possible. In most situations you just want to make simple styling changes to the controls in your application so this process should be simple. For example, if you just want to change the row color of your data grid this should be as simple as setting one property. You shouldn't have to create a full and complicated-looking template just to simply change a few colors.

How ClearStyle Works

Each key piece of the control's style is surfaced as a simple color property. This leads to a unique set of style properties for each control. For example, a **Gauge** has **PointerFill** and **PointerStroke** properties, whereas a **DataGrid** has **SelectedBrush** and **MouseOverBrush** for rows.

Let's say you have a control on your form that does not support ClearStyle. You can take the XAML resource created by ClearStyle and use it to help mold other controls on your form to match (such as grabbing exact colors). Or let's say you'd like to override part of a style set with ClearStyle (such as your own custom scrollbar). This is also possible because ClearStyle can be extended and you can override the style where desired.

ClearStyle is intended to be a solution to quick and easy style modification but you're still free to do it the old fashioned way with ComponentOne's controls to get the exact style needed. ClearStyle does not interfere with those less common situations where a full custom design is required.

Key Features

ComponentOne DataGrid for Silverlight includes several key features, such as:

- **Fully Interactive Grid**

  Enhance the end-user experience by creating a fully interactive grid. **DataGrid for Silverlight** has many built-in interactive features such as column resizing and reordering, editing, sorting, filtering, grouping, freezing, and selecting. See Run-time Interaction (page 90) for more information.

- **Data Grouping and Totals**

  DataGrid for Silverlight supports Outlook-style grouping. Simply drag a column header to the area above the grid to group the data. Expandable and collapsible nodes are automatically generated. You can also show calculated aggregate functions or totals in grouped header rows. See Grouping Columns (page 98) for details.

- **Excel-like Filtering**

  By default, **DataGrid for Silverlight** supports Excel-like filtering. This type of filtering features a drop-down menu on each column allowing users to create a filter condition. See Filtering Columns (page 95) for more information.

- **High Performance**

  DataGrid for Silverlight utilizes both row and column recycling (UI Virtualization) to achieve optimal performance when handling large data sets.

- **Several Built-in Column Types**

  DataGrid for Silverlight provides many built-in column editors that cover all of the common data types. The built-in editors include text, check box, DateTime picker, combo box and images. You can also choose from a selection of custom column editors including masked text, hyperlink, multi-line text and a color picker. See Column Types (page 59) for details.

- **RowDetails and Hierarchical Support**

  DataGrid also supports a **RowDetails** template for embedding **UIElements** inside a collapsible section of each row. For example, just embed another DataGrid and you can create a master-detail grid for displaying hierarchical data. For more information, see Adding Row Details (page 72).

- **Top and Bottom Row Templates**

  With **DataGrid for Silverlight**'s Top and Bottom row templates you can easily create and add custom rows to the grid. For example, you can design your own filter or total rows and embed any UIElements inside.

- **Multiple Selection Modes**

  Give end-users all of the following cell selection options: single cell, single row, single column, single range, multi-row, multi-column, and multi-range. With **DataGrid for Silverlight**'s clipboard support, end-users can then easily paste selected cells into any text editor, such as Microsoft Excel.

- **New Row**

  Allow users to add new rows to **DataGrid for Silverlight** by displaying an empty new row at either the top or bottom of the grid. See Adding Rows to the Grid (page 102) and Setting New Row Visibility (page 85) for details.

- **Custom Rows and Columns**

Design your own data template for each DataGrid row and create composite columns which can combine data from multiple data fields.

- **Easily Change Colors with ClearStyle**

  DataGrid for Silverlight supports ComponentOne's new ClearStyle™ technology that allows you to easily change control colors without having to change control templates. With just setting a few color properties you can quickly style the entire grid. For details, see C1DataGrid ClearStyle (page 87).

DataGrid for Silverlight Quick Start

The following quick start guide is intended to get you up and running with **ComponentOne DataGrid for Silverlight**. In this quick start you'll create a new project in Visual Studio, add **DataGrid for Silverlight** to your application, and bind it to a data source. You'll then customize the grid's appearance and behavior and run the grid application to observe run-time interactions.

Step 1 of 4: Creating a Silverlight Application

In this step you'll begin in Visual Studio to create a Silverlight grid application using **ComponentOne DataGrid for Silverlight**. You'll create a new Silverlight project and add the **C1DataGrid** control to your application.

To set up your project and add a **C1DataGrid** control to your application, complete the following steps:

1. In Visual Studio, select **File | New | Project**.
2. In the **New Project** dialog box, select a language in the left pane, and in the templates list select **Silverlight Application**. Enter a **Name** for your project and click **OK**. The **New Silverlight Application** dialog box will appear.
3. Click **OK** to accept default settings, close the **New Silverlight Application** dialog box, and create your project. The **MainPage.xaml** file should open.
4. In the Solution Explorer, right-click the project name and choose **Add Reference**. In the **Add Reference** dialog box, locate and select the **C1.Silverlight.DataGrid** assembly and click **OK** to add reference to your project.
5. Add the XAML namespace to the **UserControl** tag with the following markup: `xmlns:c1="http://schemas.componentone.com/winfx/2006/xaml"`.

   The namespaces will now appear similar to the following:
   ```
   <UserControl x:Class="C1DataGrid.MainPage"
   xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
   xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
   xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
   xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
   xmlns:c1="http://schemas.componentone.com/winfx/2006/xaml"
   mc:Ignorable="d" d:DesignHeight="300" d:DesignWidth="400">
   ```

   This is a unified namespace that will enable you to work with most ComponentOne WPF or Silverlight controls without adding multiple namespaces.
6. Add the `<c1:C1DataGrid x:Name="c1dg"></c1grid:C1DataGrid>` tag within the Grid tags on the page to add the C1DataGrid control to the application.

   The XAML will appear similar to the following:
   ```
   <Grid x:Name="LayoutRoot" Background="White">
       <c1:C1DataGrid x:Name="c1dg"></c1grid:C1DataGrid>
   </Grid>
   ```

This will add a C1DataGrid control named "c1dg" to the application. By giving the control a unique identifier, you'll be able to access the **C1DataGrid** control in code.

✓ **What You've Accomplished**

You've successfully created a basic grid application. In the next step you'll add a XML data source to your project and bind the grid to a data source.

Step 2 of 4: Binding the Grid to a Data Source

In the last step you set up the grid application, but the grid currently contains no data – if you run the application now you'll see a blank grid. In this step you'll continue in Visual Studio by adding an XML data source to your project and binding the grid to the data source.

To add a data source and bind the grid in Visual Studio, complete the following steps:

1. In the Solution Explorer window, right-click the project and select **Add | New Item**.
2. In the **Add New Item** dialog box, select **XML File** from the list of installed templates, name the file "Products.xml", and click **Add** to close the dialog box.

   The **Products.xml** file should now be included in your project, and should have opened automatically.
3. Replace the existing text in the **Products.xml** file with the following XML markup and save the file:

   - XML to add:

```
<?xml version="1.0" encoding="utf-8" ?>
<Products>
  <Product Name="Chai" Category="Beverages" Unit="10 boxes x 20 bags"
Price="18" />
  <Product Name="Chang" Category="Beverages" Unit="24 - 12 oz bottles"
Price="19" />
  <Product Name="Aniseed Syrup" Category="Condiments" Unit="12 - 550 ml
bottles" Price="10" />
  <Product Name="Chef Anton's Cajun Seasoning" Category="Condiments"
Unit="48 - 6 oz jars" Price="22" />
  <Product Name="Chef Anton's Gumbo Mix" Category="Condiments" Unit="36
boxes" Price="21.35" />
  <Product Name="Grandma's Boysenberry Spread" Category="Condiments"
Unit="12 - 8 oz jars" Price="25" />
  <Product Name="Uncle Bob's Organic Dried Pears" Category="Produce"
Unit="12 - 1 lb pkgs." Price="30" />
  <Product Name="Northwoods Cranberry Sauce" Category="Condiments"
Unit="12 - 12 oz jars" Price="40" />
  <Product Name="Mishi Kobe Niku" Category="Meat/Poultry" Unit="18 -
500 g pkgs." Price="97" />
  <Product Name="Ikura" Category="Seafood" Unit="12 - 200 ml jars"
Price="31" />
  <Product Name="Queso Cabrales" Category="Dairy Products" Unit="1 kg
pkg." Price="21" />
  <Product Name="Queso Manchego La Pastora" Category="Dairy Products"
Unit="10 - 500 g pkgs." Price="38" />
  <Product Name="Konbu" Category="Seafood" Unit="2 kg box" Price="6" />
  <Product Name="Tofu" Category="Produce" Unit="40 - 100 g pkgs."
Price="23.25" />
  <Product Name="Genen Shouyu" Category="Condiments" Unit="24 - 250 ml
bottles" Price="15.5" />
  <Product Name="Pavlova" Category="Condiments" Unit="32 - 500 g boxes"
Price="17.45" />
```

```
  <Product Name="Alice Mutton" Category="Meat/Poultry" Unit="20 - 1 kg
tins" Price="39" />
  <Product Name="Carnarvon Tigers" Category="Seafood" Unit="16 kg pkg."
Price="62.5" />
  <Product Name="Teatime Chocolate Biscuits" Category="Confections"
Unit="10 boxes x 12 pieces" Price="9.2" />
  <Product Name="Sir Rodney's Marmalade" Category="Confections"
Unit="30 gift boxes" Price="81" />
  <Product Name="Sir Rodney's Scones" Category="Confections" Unit="24
pkgs. x 4 pieces" Price="10" />
  <Product Name="Gustaf's Knäckebröd" Category="Grains/Cereals"
Unit="24 - 500 g pkgs." Price="21" />
  <Product Name="Tunnbröd" Category="Grains/Cereals" Unit="12 - 250 g
pkgs." Price="9" />
  <Product Name="Guaraná Fantástica" Category="Beverages" Unit="12 -
355 ml cans" Price="4.5" />
  <Product Name="NuNuCa Nuß-Nougat-Creme" Category="Confections"
Unit="20 - 450 g glasses" Price="14" />
</Products>
```

This will add data taken from the *Products* table of the standard Microsoft Northwind database.

4. Right-click the project and select **Add Reference**. In the **Add Reference** dialog box, locate **System.Xml.Linq** and click **OK** to add the reference.
5. Choose **MainPage.xaml**, right-click the page, and select **View Code** in the context menu to open the Code Editor.
6. At the top of the Code Editor, add the following code to import namespaces:

   - Visual Basic

```
Imports System.Xml.Linq
Imports C1.Silverlight.DataGrid
```

   - C#

```
using System.Xml.Linq;
using C1.Silverlight.DataGrid;
```

7. Replace the existing code with the following code to initialize the data source, and bind the **C1DataGrid.ItemsSource** property to the XML data source:

   - Visual Basic

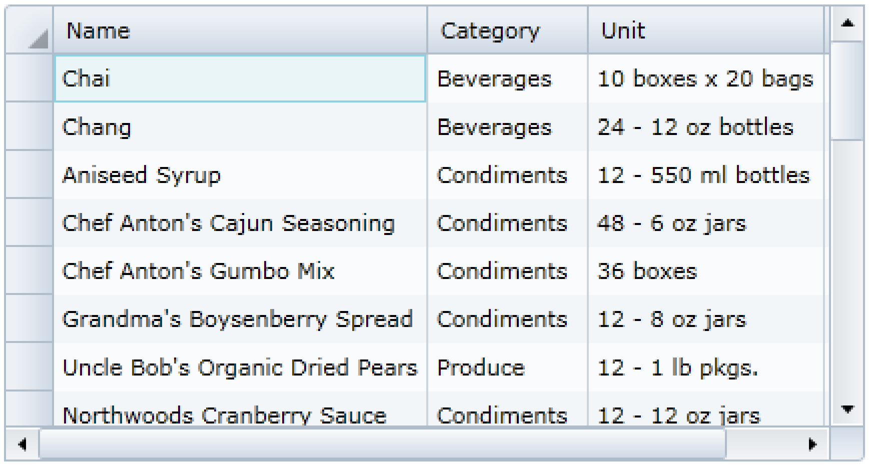
```
Partial Public Class MainPage
    Inherits UserControl
    Public Sub New()
        InitializeComponent()
        LoadData()
    End Sub
    ' Create the Product class.
    Public Class Product
        Private _Name As String
        Public Property Name() As String
            Get
                Return _Name
            End Get
            Set(ByVal value As String)
                _Name = value
            End Set
        End Property
        Private _Category As String
        Public Property Category() As String
            Get
                Return _Category
            End Get
            Set(ByVal value As String)
                _Category = value
            End Set
        End Property
        Private _Unit As String
        Public Property Unit() As String
            Get
                Return _Unit
            End Get
            Set(ByVal value As String)
                _Unit = value
            End Set
        End Property
        Private _Price As String
        Public Property Price() As String
            Get
                Return _Price
            End Get
            Set(ByVal value As String)
                _Price = value
            End Set
        End Property
    End Class
    Private Sub LoadData()
        ' Initialize the XML data source.
        Dim ProductsDoc As XDocument = XDocument.Load("Products.xml")
        Dim data As IEnumerable(Of Product) = (From Product In
ProductsDoc.Descendants("Product") Select New Product With {.Name =
Product.Attribute("Name").Value, .Category =
Product.Attribute("Category").Value, .Unit =
Product.Attribute("Unit").Value, .Price =
Product.Attribute("Price").Value}).ToList
        ' Bind the C1DataGrid control to the data source.
        c1dg.ItemsSource = data
    End Sub
End Class
```

- C#

```
public MainPage()
{
    InitializeComponent();
    LoadData();
}
// Create the Product class.
public class Product
{
    public string Name { get; set; }
    public string Category { get; set; }
    public string Unit { get; set; }
    public string Price { get; set; }
}
private void LoadData()
{
    // Initialize the XML data source.
    XDocument ProductsDoc = XDocument.Load("Products.xml");
    List<Product> data = ( from Product in ProductsDoc.Descendants(
"Product" )
        select new Product
        {
            Name = Product.Attribute("Name").Value,
            Category = Product.Attribute("Category").Value,
            Unit = Product.Attribute("Unit").Value,
            Price = Product.Attribute("Price").Value
         }
    ).ToList();
    // Bind the C1DataGrid control to the data source.
    c1dg.ItemsSource = data;
}
```

What You've Accomplished

If you save and run your application you'll observe that the grid is now populated with data from the Products.xml file:



Name	Category	Unit
Chai	Beverages	10 boxes x 20 bags
Chang	Beverages	24 - 12 oz bottles
Aniseed Syrup	Condiments	12 - 550 ml bottles
Chef Anton's Cajun Seasoning	Condiments	48 - 6 oz jars
Chef Anton's Gumbo Mix	Condiments	36 boxes
Grandma's Boysenberry Spread	Condiments	12 - 8 oz jars
Uncle Bob's Organic Dried Pears	Produce	12 - 1 lb pkgs.
Northwoods Cranberry Sauce	Condiments	12 - 12 oz jars

You've successfully bound **DataGrid for Silverlight**'s **C1DataGrid** control to an XML data source. In the next step you'll customize the appearance and behavior of the **C1DataGrid** control.

Step 3 of 4: Customizing the Grid's Appearance and Behavior

In the previous steps you worked in Visual Studio to create a new Silverlight project and bind **DataGrid for Silverlight** to a data source. In this step you'll customize the grid application's appearance and behavior.

To customize **DataGrid for Silverlight**, complete the following steps:

1. Return to the **MainPage.xaml** file. In this example you'll be customizing the grid in XAML rather than code.
2. Locate the `<c1grid:C1DataGrid>` tag in the XAML window and add `CanUserGroup="True"` to it. This will enable the grouping area of the grid.

   The XAML will now look similar to the following:
   ```
   <c1grid:C1DataGrid x:Name="c1dg" CanUserGroup="True">
   ```

3. Add `NewRowVisibility="Top"` to the `<c1grid:C1DataGrid>` tag in the XAML window. This will move the add new row to the top of the grid.

   The XAML will now look similar to the following:
   ```
   <c1grid:C1DataGrid x:Name="c1dg" CanUserGroup="True"
   NewRowVisibility="Top">
   ```

4. Add `VerticalScrollBarVisibility="Visible"` `HorizontalScrollBarVisibility="Visible"` to the `<c1grid:C1DataGrid>` tag. This will ensure that the horizontal and vertical scroll bars are always visible.

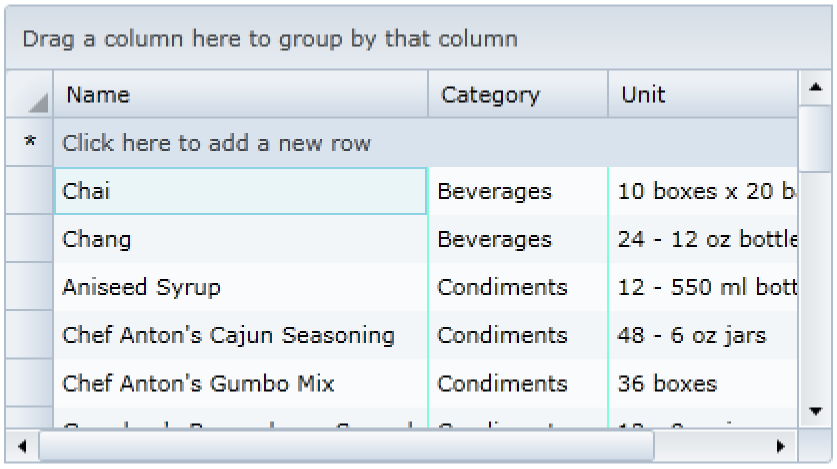
   The XAML will now look similar to the following:
   ```
   <c1grid:C1DataGrid x:Name="c1dg" CanUserGroup="True"
   NewRowVisibility="Top" VerticalScrollBarVisibility="Visible"
   HorizontalScrollBarVisibility="Visible">
   ```

5. Add `VerticalGridLinesBrush="Aquamarine"` to the `<c1grid:C1DataGrid>` tag. This changes the color of the vertical grid lines.

   The XAML will now look similar to the following:
   ```
   <c1grid:C1DataGrid x:Name="c1dg" CanUserGroup="True"
   NewRowVisibility="Top" VerticalScrollBarVisibility="Visible"
   HorizontalScrollBarVisibility="Visible"
   VerticalGridLinesBrush="Aquamarine">
   ```

What You've Accomplished

Save and run your application and observe that you've changed the appearance of the grid and the columns that are displayed:



Drag a column here to group by that column

	Name	Category	Unit
*	Click here to add a new row		
	Chai	Beverages	10 boxes x 20 b
	Chang	Beverages	24 - 12 oz bottle
	Aniseed Syrup	Condiments	12 - 550 ml bott
	Chef Anton's Cajun Seasoning	Condiments	48 - 6 oz jars
	Chef Anton's Gumbo Mix	Condiments	36 boxes

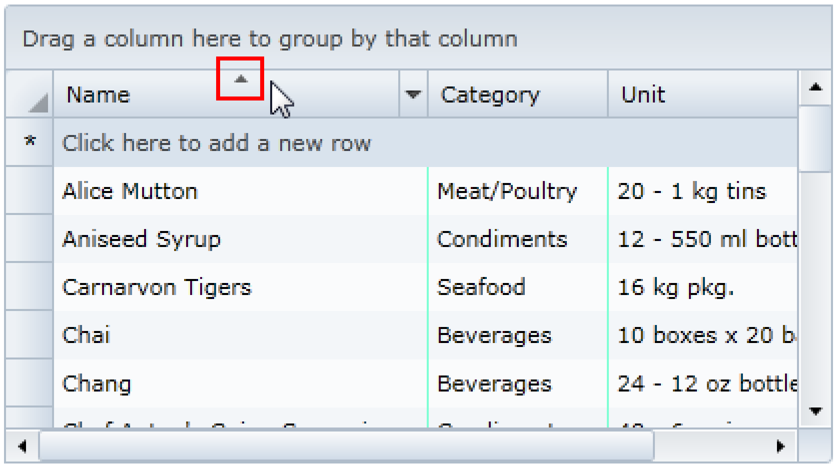
You've successfully customized the appearance and behavior of your grid. In the next step you'll explore some of the run-time interactions that are possible in your grid application.

Step 4 of 4: Running the Grid Application

Now that you've created a grid application, bound the grid to a data source, and customized the grid's appearance and behavior, the only thing left to do is run your application. To observe **ComponentOne DataGrid for Silverlight**'s run-time interactions, complete the following steps:

1. Select **Debug | Start Debugging** to run your application.

2. Click the **Name** header once to sort the grid by product name. Notice that a sort indicator glyph appears to indicate the column being sorted and the direction of the sort.

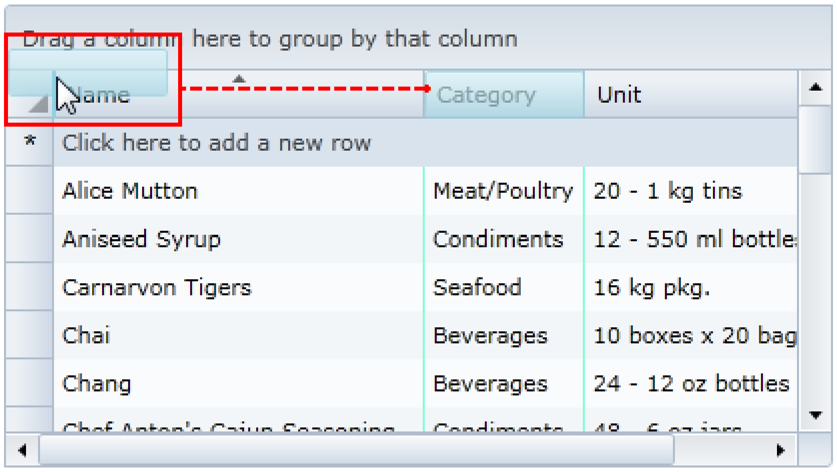


Drag a column here to group by that column

	Name	Category	Unit
*	Click here to add a new row		
	Alice Mutton	Meat/Poultry	20 - 1 kg tins
	Aniseed Syrup	Condiments	12 - 550 ml bott
	Carnarvon Tigers	Seafood	16 kg pkg.
	Chai	Beverages	10 boxes x 20 b
	Chang	Beverages	24 - 12 oz bottle

You can sort multiple columns by holding the CTRL key while clicking another column.

3. Re-order the columns by clicking the **Category** column header and dragging it in front of the **Name** column header:

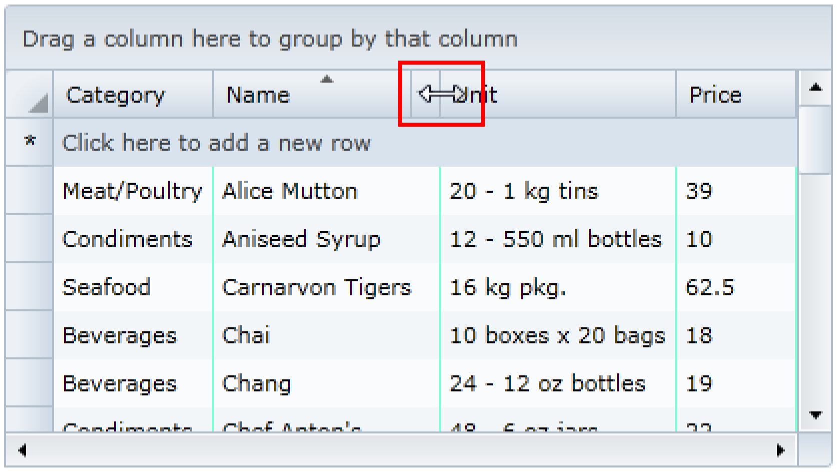


Drag a column here to group by that column

	Name	Category	Unit
*	Click here to add a new row		
	Alice Mutton	Meat/Poultry	20 - 1 kg tins
	Aniseed Syrup	Condiments	12 - 550 ml bottle
	Carnarvon Tigers	Seafood	16 kg pkg.
	Chai	Beverages	10 boxes x 20 bag
	Chang	Beverages	24 - 12 oz bottles
	Chef Anton's Cajun Seasoning	Condiments	48 - 6 oz jars

The *Category* column will now appear before the *Name* column.

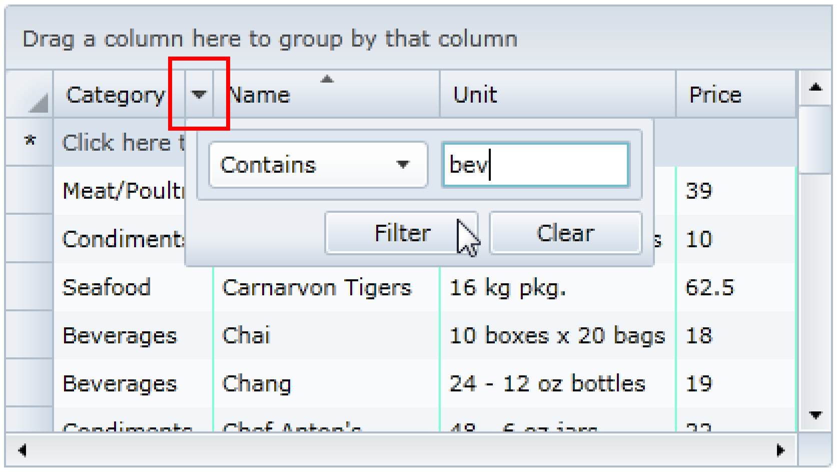
4. Resize a column, here the *Name* column, by clicking the right edge of the column and dragging the edge to a new location.



Drag a column here to group by that column

Category	Name	Unit	Price
Click here to add a new row			
Meat/Poultry	Alice Mutton	20 - 1 kg tins	39
Condiments	Aniseed Syrup	12 - 550 ml bottles	10
Seafood	Carnarvon Tigers	16 kg pkg.	62.5
Beverages	Chai	10 boxes x 20 bags	18
Beverages	Chang	24 - 12 oz bottles	19
Condiments	Chef Anton's	48 - 6 oz jars	22

5. Filter the content of a column by clicking the drop-down arrow in the *Category* header, entering "bev" in the filter text box, and pressing the **Filter** button, so that only items beginning with that string appear:

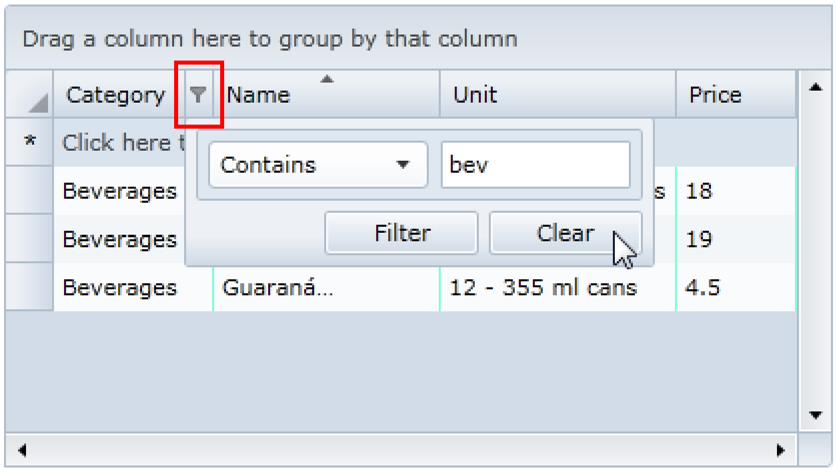


Drag a column here to group by that column

Category	Name	Unit	Price
Click here t			
Meat/Poult			39
Condiment		s	10
Seafood	Carnarvon Tigers	16 kg pkg.	62.5
Beverages	Chai	10 boxes x 20 bags	18
Beverages	Chang	24 - 12 oz bottles	19
Condiments	Chef Anton's	48 - 6 oz jars	22

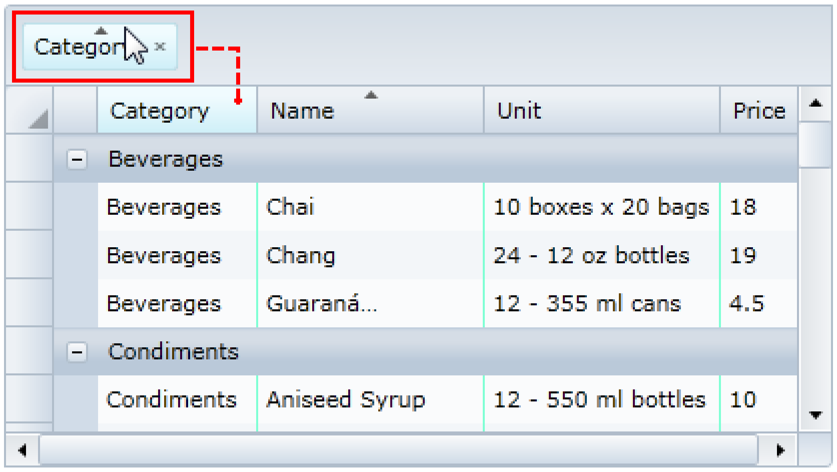
Notice that the drop-down arrow icon is replaced with a filter icon, indicating that the column is filtered. Note that the vertical scroll bar remains visible, though content cannot be scrolled vertically. This is because you set the **VerticalScrollBarVisibility** property to **Visible**.

6. Clear the filter by clicking the **Filter** icon in the *Category* column header and clicking the **Clear** button:



Category	Name	Unit	Price
Beverages			18
Beverages			19
Beverages	Guaraná...	12 - 355 ml cans	4.5

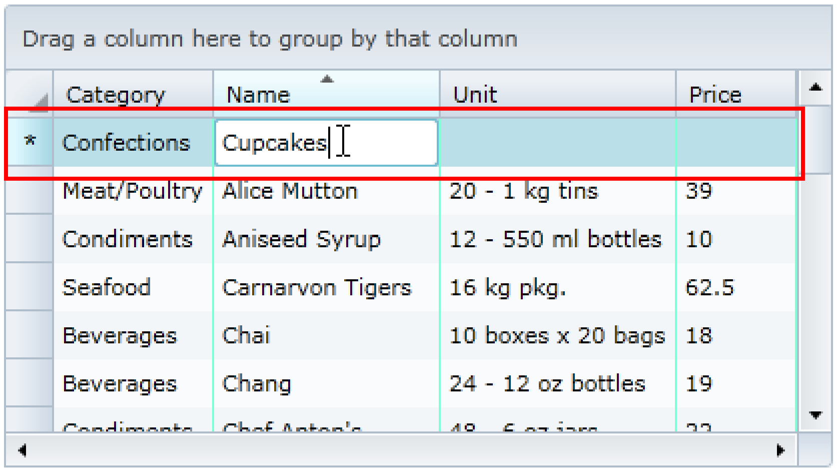
7. Drag the **Category** header to the grouping area to group the grid by category:



Category	Name	Unit	Price
Beverages			
Beverages	Chai	10 boxes x 20 bags	18
Beverages	Chang	24 - 12 oz bottles	19
Beverages	Guaraná...	12 - 355 ml cans	4.5
Condiments			
Condiments	Aniseed Syrup	12 - 550 ml bottles	10

You can repeat this with additional columns to group by multiple criteria.

8. Click the **X** button in the *Category* column header in the grouping area to no longer group columns by category.
9. Add new data to the grid by clicking in the new row and typing content:

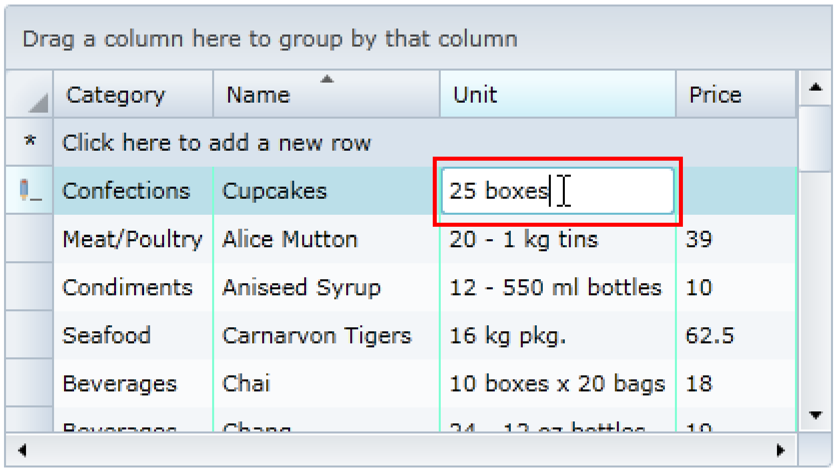


Drag a column here to group by that column

	Category	Name	Unit	Price
*	Confections	Cupcakes		
	Meat/Poultry	Alice Mutton	20 - 1 kg tins	39
	Condiments	Aniseed Syrup	12 - 550 ml bottles	10
	Seafood	Carnarvon Tigers	16 kg pkg.	62.5
	Beverages	Chai	10 boxes x 20 bags	18
	Beverages	Chang	24 - 12 oz bottles	19

Click away from the row or press ENTER for the content you added to be included in the grid.

10. To edit the contents of a cell, click once on a cell, type text to add or change content, and press the ENTER key:



Drag a column here to group by that column

	Category	Name	Unit	Price
*	Click here to add a new row			
	Confections	Cupcakes	25 boxes	
	Meat/Poultry	Alice Mutton	20 - 1 kg tins	39
	Condiments	Aniseed Syrup	12 - 550 ml bottles	10
	Seafood	Carnarvon Tigers	16 kg pkg.	62.5
	Beverages	Chai	10 boxes x 20 bags	18

Notice that a pencil icon appears in the row indicator column, specifying that a cell in that row is in edit mode.

Congratulations! You've completed the **DataGrid for Silverlight** quick start and created a **DataGrid for Silverlight** grid application, bound the grid to a data source, customized the appearance of the grid, and viewed some of the run-time capabilities of your grid application.

Working with DataGrid for Silverlight

ComponentOne DataGrid for Silverlight allows you to select, edit, add, delete, filter, group, and sort the items displayed in the table generated by the **C1DataGrid** component.

The columns of a table created using the **C1DataGrid** component correspond to the fields in a data source. You can control which columns are displayed, the types of columns to display, and the appearance of the whole table.

Using the **AutoGenerateColumns** property, you can generate columns automatically, manually, or both. Setting this property to **True** (default) creates columns automatically when the **ItemsSource** property is set. Setting this property to **False** allows you to specify the columns to display, which are added to the **Columns** collection.

Note: By default explicitly declared columns are rendered first, followed by automatically generated columns. You can change the order of rendered columns by setting the **DisplayIndex** property of the column. Automatically generated columns are now added to the **Columns** collection.

Class Hierarchy

The following list summarizes the class relationships between the more important classes included in the **DataGrid for Silverlight**:

- C1.Silverlight.DataGrid.C1DataGrid : System.Windows.Controls.Control
  Encapsulates most of the grid functionality. This component is shown in Visual Studio's Toolbox.
- C1.Silverlight.DataGrid.DataGridColumn : System.Object
  Represents a column in the grid.
- C1.Silverlight.DataGridColumnCollection : System.Object
  Represents the collection of columns of the data grid.
- C1.Silverlight.DataGrid.DataGridColumnHeaderPresenter : System.Windows.Controls.Control
  Content control that represent the header of a column; this control contains the sort, resize and filter elements.
- C1.Silverlight.DataGrid.DataGridRow : System.Object
  Represents a row in the grid.
- C1.Silverlight.DataGrid.DataGridRowCollection : System.Object
  Collection of rows.
- C1.Silverlight.DataGrid.DataGridCell : System.Object
  Represents an individual grid cell.

Data Binding

ComponentOne DataGrid for Silverlight's **C1DataGrid** control can be bound to any object that implements the **System.Collections.IEnumerable** interface (such as **XmlDataProvider**, **ObjectDataProvider**, **DataSet**, **DataView**, and so on). You can use the **C1DataGrid.ItemsSource** property to bind the **C1DataGrid**.

To bind the grid, simply set the **ItemsSource** property to an **IEnumerable** implementation. Each row in the data grid will be bound to an object in the data source, and each column in the data grid will be bound to a property of the data object.

Note that in order for the **C1DataGrid** user interface to update automatically when items are added to or removed from the source data, the control must be bound to a collection that implements **INotifyCollectionChanged**, such as an **ObservableCollection<(Of <(T>)>)**.

See WCF RIA Services Data Binding (page 58) and Binding the Grid to a WCF RIA Services Data Source (page 126) for information about binding the grid to an RIA Services data source. See Binding the Grid to an RSS Feed (page 112) and Binding the Grid to a Web Service (page 105) for data binding examples. For steps on binding a **C1DataGrid** control to an XML data source, see the DataGrid for Silverlight Quick Start (page 46).

WCF RIA Services Data Binding

ComponentOne DataGrid for Silverlight's **C1DataGrid** control support direct binding to the WCF RIA services **DomainDataSoure**. There are two ways of doing so codelessly in XAML. You can bind directly to the **DomainDataSource** which works but involve some loss in filtering functionality, or you can use an **Adaptor** class to bind the grid.

You can bind the grid directly to the **DomainDataSource**, using XAML markup similar to the following:

```
<ria:DomainDataSource x:Name="_myDataSource" QueryName="GetProducts"
PageSize="8">

                <ria:DomainDataSource.DomainContext>

                    <local:NorthwindContext/>

                </ria:DomainDataSource.DomainContext>

</ria:DomainDataSource>


<c1data:C1DataGrid x:Name="_dataGrid" ItemsSource="{Binding Data,
ElementName=_myDataSource}">
```

You can bind the grid this way and it will work, but you will lose **C1DataGrid**'s built-in filtering functionality because RIA services use a different filtering approach than standard **CollectionView**.

To retain all functionality you will need an additional class to "translate" the filtering information so RIA services can perform filtering; that's when the C1RiaAdapter class comes into play. It performs the translations required for the C1DataGrid filtering to work with RIA.

You can use XAML markup similar to the following:

Indirect binding to **DomainDataSource** (binding through **Adapter** class)

```
<adapter:C1RiaAdapter x:Name="_adapter" DataGrid="{Binding
ElementName=_dataGrid}">

            <ria:DomainDataSource x:Name="_myDataSource"
QueryName="GetProducts" PageSize="8">

                <ria:DomainDataSource.DomainContext>

                    <local:NorthwindContext/>

                </ria:DomainDataSource.DomainContext>

            </ria:DomainDataSource>

</adapter:C1RiaAdapter>



<c1data:C1DataGrid x:Name="_dataGrid" ItemsSource="{Binding Data,
ElementName=_adapter}">
```

If you bind the grid this way, you will get filtering support out of the box. Of course, if you do not need filtering support, you can always bind the grid directly without using **C1RiaAdapter**.

For an example, see the Binding the Grid to a WCF RIA Services Data Source (page 126) topic.

Defining Columns

You can use **ComponentOne DataGrid for Silverlight**'s **Columns** collection to programmatically add, insert, remove, and change any columns in the control at run time. You can also specify columns in XAML with or without automatically generating columns.

Creating your own columns enables you to use additional column types, such as the **DataGridTemplateColumn** type or custom column types. The **DataGridTemplateColumn** type provides an easy way to create a simple custom column. The **CellTemplate** and **CellEditingTemplate** properties enable you to specify content templates for both display and editing modes.

Generating Columns

By default, the **C1DataGrid** control generates columns automatically, based on the type of data, when you set the **ItemsSource** property. The generated columns are of type **DataGridCheckBoxColumn** for bound Boolean (and nullable Boolean) properties, and of type **DataGridTextColumn** for bound string data, **DataGridComboBoxColumn** for bound enum data, **DataGridDateTimeColumn** for bound date/time data, and **DataGridNumericColumn** for bound numeric data. Bound undefined data is displayed in a **DataGridBoundColumn** type column. If a property does not have a String or numeric value type, the generated text box columns are read-only and display the data object's **ToString** value.

You can prevent automatic column generation by setting the **AutoGenerateColumns** property to **False**. This is useful if you want to create and configure all columns explicitly. Alternatively, you can let the data grid generate columns, but handle the **AutoGeneratingColumn** event to customize columns after creation. To rearrange the display order of the columns, you can set the **DisplayIndex** property for individual columns.

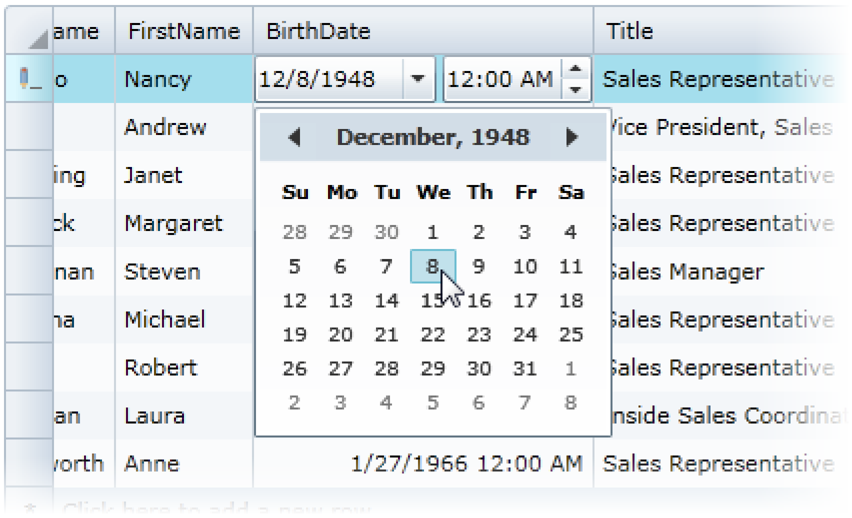
Column Types

ComponentOne DataGrid for Silverlight's **C1DataGrid** control provides a flexible way to display a collection of data in rows and columns by providing many built-in column editors that cover all of the common data types. Built-in column types include:

Column Type	Description
DataGridBoundColumn	A column that can bind to a property in the grid's data source. This is the default column type for bound undefined data.
DataGridTextColumn	A text column. This is the default column type for bound string data.
DataGridCheckBoxColumn	A check box column. This is the default column type for bound Boolean data.
DataGridComboBoxColumn	A combo box column. This is the default column type for bound enumeration type data.
DataGridDateTimeColumn	A date time column (see below for an image). This is the default column type for bound date/time data.
DataGridImageColumn	An image column.
DataGridNumericColumn	A numeric column. This is the default column type for bound numeric data (the format will be inferred from the type. For example, if the type is int the format will not contain decimal places).
DataGridTemplateColumn	A template column for hosting custom content.
CustomColumns	A custom column. See the C1DataGrid_Demo sample for examples ofcustom columns like a Composite Column, Color Column, Gif Column, Hyperlink Column, Masked Text Column, Multi line Text

	Column, and so on.

These column types can provide built-in input validation; for example the **DataGridDateTimeColumn** column includes a calendar for selecting a date:



ame	FirstName	BirthDate	Title
o	Nancy	12/8/1948 12:00 AM	Sales Representative
	Andrew		ice President, Sales
ing	Janet		ales Representative
ck	Margaret		ales Representative
nan	Steven		ales Manager
na	Michael		ales Representative
	Robert		ales Representative
an	Laura		nside Sales Coordina
orth	Anne	1/27/1966 12:00 AM	Sales Representative

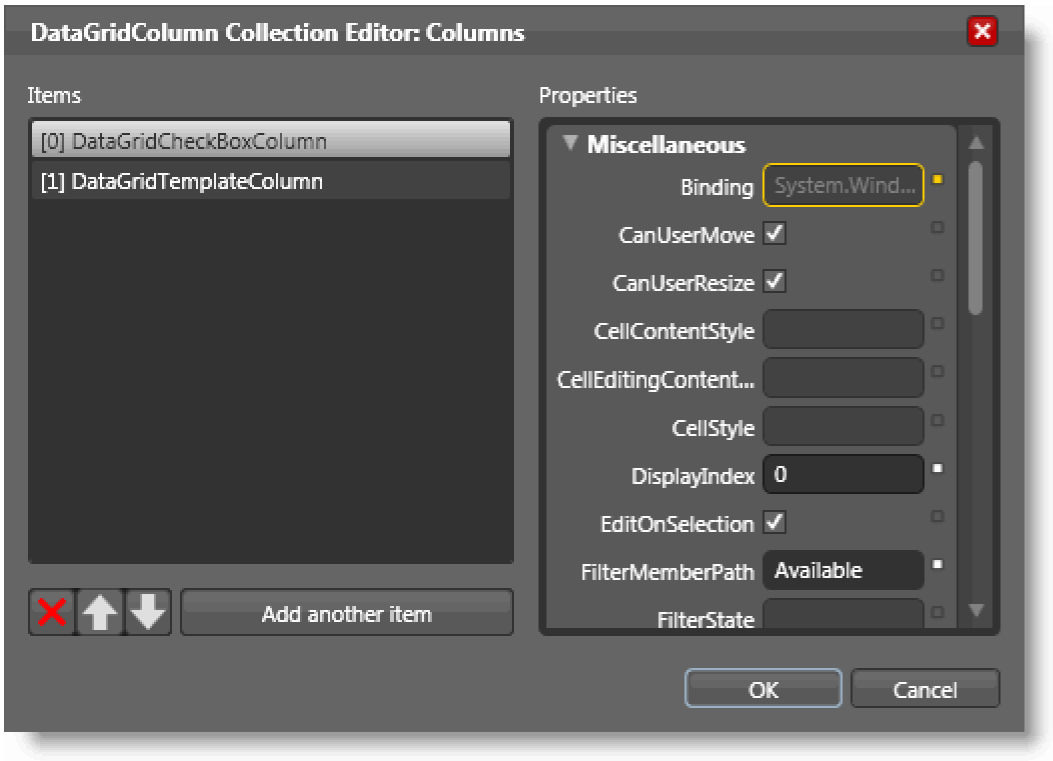
December, 1948

Su	Mo	Tu	We	Th	Fr	Sa
28	29	30	1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30	31	1
2	3	4	5	6	7	8

Explicitly Defining Columns

If you choose, you can explicitly define columns. If the **AutoGenerateColumns** property is **False** only the columns you have defined will appear in the grid.

In Microsoft Expression Blend, you can use the **DataGridColumn Collection Editor** to define columns in your grid. Select the **C1DataGrid** control, and in the Properties window select the ellipsis button next to the **Columns(Collection)** item in the **Miscellaneous** group. The **DataGridColumn Collection Editor** dialog box will appear:


DataGridColumn Collection Editor: Columns

Items

[0] DataGridCheckBoxColumn
[1] DataGridTemplateColumn

Properties

Miscellaneous

Binding	System.Wind...
CanUserMove	✓
CanUserResize	✓
CellContentStyle	
CellEditingContent...	
CellStyle	
DisplayIndex	0
EditOnSelection	✓
FilterMemberPath	Available
FilterState	

You can also define custom columns in the grid in XAML by using the **Columns** collection.

For example:

- XAML

```
<c1:C1DataGrid x:Name="grid" Grid.Row="1" Grid.ColumnSpan="2" Margin="5"
AutoGeneratingColumn="grid_AutoGeneratingColumn" CanUserAddRows="False"
ColumnHeaderHeight="30" >
    <c1:C1DataGrid.Columns>
        <!--
        Custom check box column.
        Adds a check box to the header and listens to its events.
        -->
        <c1:DataGridCheckBoxColumn Binding="{Binding Available,
Mode=TwoWay}" DisplayIndex="0" SortMemberPath="Available"
FilterMemberPath="Available" MinWidth="108" >
            <c1:DataGridColumn.Header>
                <StackPanel Orientation="Horizontal"
HorizontalAlignment="Left">
                    <TextBlock Margin="6,0,6,0" VerticalAlignment="Center"
Text="Available"/>
                    <CheckBox HorizontalAlignment="Left"
IsHitTestVisible="True" VerticalAlignment="Center" Grid.Column="1"
Checked="CheckBox_Checked" Unchecked="CheckBox_Checked"
Loaded="CheckBox_Loaded"/>
                </StackPanel>
            </c1:DataGridColumn.Header>
        </c1:DataGridCheckBoxColumn>
        <!--
        Custom "merged" column made with a DataGridTemplateColumn.
        You can also inherit from DataGridTemplateColumn and set
        this configuration in the constructor to make your XAML
        cleaner.
        -->
        <c1:DataGridTemplateColumn>
            <c1:DataGridTemplateColumn.Header>
                <local:MergedColumnEditor ControlMode="Header" />
            </c1:DataGridTemplateColumn.Header>
            <c1:DataGridTemplateColumn.CellTemplate>
                <DataTemplate>
                    <local:MergedColumnEditor ControlMode="Cell" />
                </DataTemplate>
            </c1:DataGridTemplateColumn.CellTemplate>
            <c1:DataGridTemplateColumn.CellEditingTemplate>
                <DataTemplate>
                    <local:MergedColumnEditor ControlMode="EditingCell" />
                </DataTemplate>
            </c1:DataGridTemplateColumn.CellEditingTemplate>
        </c1:DataGridTemplateColumn>
    </c1:C1DataGrid.Columns>
</c1:C1DataGrid>
```

Customizing Automatically Generated Columns

You can customize columns even if columns are automatically generated. If the **AutoGenerateColumns** property is set to **True** and columns are automatically generated, you can customize how generated columns are displayed in code by handling the **C1DataGrid.AutoGeneratingColumn** event.

Adding the AutoGeneratingColumn Event Handler

Complete the following steps to add the **AutoGeneratingColumn** event handler:

1. Switch to Code view and add an event handler for the **AutoGeneratingColumn** event, for example:

   - Visual Basic

   ```
   Private Sub C1DataGrid1_AutoGeneratingColumn(ByVal sender As
   System.Object, ByVal e As
   C1.Silverlight.DataGrid.DataGridAutoGeneratingColumnEventArgs) Handles
   C1DataGrid1.AutoGeneratingColumn
       ' Add code here.
   End Sub
   ```

   - C#

   ```
   private void C1DataGrid1_AutoGeneratingColumn(object sender,
   C1.Silverlight.DataGrid.DataGridAutoGeneratingColumnEventArgs e)
   {
       // Add code here.
   }
   ```

2. Switch to Source view and add the event handler to instances of the **C1DataGrid** control, for example:

   ```
   <c1:C1DataGrid x:Name="c1DataGrid1" AutoGenerateColumns="True"
   AutoGeneratingColumn=" c1DataGrid1_AutoGeneratingColumn"></c1:C1DataGrid>
   ```

You can now add code to the **AutoGeneratingColumn** event handler to customize the appearance and behavior of automatically generated columns. Below are examples of customizing column formatting and behavior.

Canceling Column Generation

You can cancel the generation of specific columns in the **AutoGeneratingColumn** event. For example, you can use the following code to cancel the generation of Boolean columns in the grid:

- Visual Basic

```
Private Sub C1DataGrid1_AutoGeneratingColumn(ByVal sender As
System.Object, ByVal e As
C1.Silverlight.DataGrid.DataGridAutoGeneratingColumnEventArgs) Handles
C1DataGrid1.AutoGeneratingColumn
    ' Cancel automatic generation of all Boolean columns.
    If e.Property.PropertyType Is GetType(Boolean) Then
        e.Cancel = True
    End If
End Sub
```

- C#

```
private void c1DataGrid1_AutoGeneratingColumn(object sender,
C1.Silverlight.DataGrid.DataGridAutoGeneratingColumnEventArgs e)
{
    // Cancel automatic generation of all Boolean columns.
    if (e.Property.PropertyType == typeof(bool))
        e.Cancel = true;
}
```

Changing a Column Header

In the **AutoGeneratingColumn** event you can change the text that appears in the header of automatically generated columns. For example, you can change the "ProductName" column so that it appears with the "Name" header using the following code:

- Visual Basic

```
Private Sub C1DataGrid1_AutoGeneratingColumn(ByVal sender As
System.Object, ByVal e As
C1.Silverlight.DataGrid.DataGridAutoGeneratingColumnEventArgs) Handles
C1DataGrid1.AutoGeneratingColumn
    ' Modify the header of the ProductName column.
    If e.Column.Header.ToString() = "ProductName" Then
         e.Header = "Name"
    End If
End Sub
```

- C#

```
private void c1DataGrid1_AutoGeneratingColumn(object sender,
C1.Silverlight.DataGrid.DataGridAutoGeneratingColumnEventArgs e)
{
    // Modify the header of the ProductName column.
    if (e.Column.Header.ToString() == "ProductName")
        e.Column.Header = "Name";
}
```

Preventing Column Interaction

Using the **AutoGeneratingColumn** event you can change how end users interact with specific generated columns. For example, you can prevent users from moving read-only columns with the following code:

- Visual Basic

```
Private Sub C1DataGrid1_AutoGeneratingColumn(ByVal sender As
System.Object, ByVal e As
C1.Silverlight.DataGrid.DataGridAutoGeneratingColumnEventArgs) Handles
C1DataGrid1.AutoGeneratingColumn
    ' Modify the header of the ProductName column.
    If e.Column.IsReadOnly = True Then
       e.Column.CanUserMove = False
    End If
End Sub
```

- C#

```
private void c1DataGrid1_AutoGeneratingColumn(object sender,
C1.Silverlight.DataGrid.DataGridAutoGeneratingColumnEventArgs e)
{
    // Modify the header of the ProductName column.
    if (e.Column.IsReadOnly == true)
        e.Column.CanUserMove = false;
}
```

Creating Custom Columns

ComponentOne DataGrid for Silverlight supports creating specific behavior columns. For example you can create a Hyperlink column, a GIF column, a Rich Text column, and so on.

By creating a custom column you'll be able to customize the cell content and editing content of all the cells belonging to a column, you can even customize the header presenter of the column.

First, you should add a new class file where the custom column will be written, for example complete the following steps:

1. Navigate to the Solution Explorer, right-click the project name and select **Add | New Item**.
2. In the **Add New Item** dialog box choose **Class** in the list of templates.
3. Name the class, for example "DataGridHyperlinkColumn", and click the **Add** button to add the class to the project.

Once the file is created it must inherit from **DataGridBoundColumn**. Update the class so it appears similar to the following:

- Visual Basic

```
Imports C1.Silverlight.DataGrid
Public Class DataGridHyperlinkColumn
    Inherits DataGridBoundColumn
End Class
```

- C#

```
using C1.Silverlight.DataGrid;
public class DataGridHyperlinkColumn : DataGridBoundColumn
{

}
```

Customizing Column Cell Content

In this section you'll find information about changing the UI element shown as the content of cells belonging to a column when the cell is not in editing mode.

It's important to note that cell content UI elements are recycled by the data grid; that means that this column could potentially use UI elements created by other columns.

To implement custom cell content you'll need to override the following methods:

- **GetCellContentRecyclingKey**: Key used to store the cell content for future reuse in a shared pool. Columns returning the same **RecyclingKey** will be candidates to share the same cell content instances.
- **CreateCellContent**: Creates the visual element that will be used to display the information inside a cell.
- **BindCellContent**: Initializes the cell content presenter. This method must set **cellContent** properties, the **SetBinding** of the corresponding dependency property being "row.DataItem", the source which can be set directly in the binding or in the **DataContext** of the **cellContent**.

- **UnbindCellContent**: This method is called before the cell content is recycled.

In the implementation of a hyperlink column the methods might look similar to the example below. In the following method a different key for this column is returned (the default key is **typeof(TextBlock)**), That means this column will not share the cell content element with other columns (unless it would be another column which returned the same key, but that's not likely to happen).

- Visual Basic

```
Public Overloads Overrides Function GetCellContentRecyclingKey(ByVal row
As DataGridRow) As Object
    Return (GetType(HyperlinkButton))
End Function
```

- C#

```
public override object GetCellContentRecyclingKey(DataGridRow row)
{
    return typeof(HyperlinkButton);
}
```

The **CreateCellContent** method will be called by the data grid if there is no recycled hyperlink. In this case a new hyperlink will be created which will be used in the cell once the cell that contains the hyperlink is unloaded; the hyperlink will be saved to be used in a future cell:

- Visual Basic

```
Public Overloads Overrides Function CreateCellContent(ByVal row As
DataGridRow) As FrameworkElement
    Return New HyperlinkButton()
End Function
```

- C#

```
public override FrameworkElement CreateCellContent(DataGridRow row)
{
    return new HyperlinkButton();
}
```

After the hyperlink is created or a recycled one is taken, the **BindCellContent** method will be called by the data grid passing the hyperlink as a parameter. In this method you should set the properties of the hyperlink to bind it to the data of the cell:

- Visual Basic

```
Public Overloads Overrides Sub BindCellContent(ByVal cellContent As
FrameworkElement, ByVal row As DataGridRow)
    Dim hyperlink = DirectCast(cellContent, HyperlinkButton)
    If Binding IsNot Nothing Then
        Dim newBinding As Binding = CopyBinding(Binding)
        newBinding.Source = row.DataItem
        hyperlink.SetBinding(HyperlinkButton.NavigateUriProperty,
newBinding)
    End If
    hyperlink.HorizontalAlignment = HorizontalAlignment
    hyperlink.VerticalAlignment = VerticalAlignment
End Sub
```

- C#

```
public override void BindCellContent(FrameworkElement cellContent,
DataGridRow row)
{
    var hyperlink = (HyperlinkButton)cellContent;
    if (Binding != null)
    {
        Binding newBinding = CopyBinding(Binding);
        newBinding.Source = row.DataItem;
        hyperlink.SetBinding(HyperlinkButton.NavigateUriProperty, 
newBinding);
    }
    hyperlink.HorizontalAlignment = HorizontalAlignment;
    hyperlink.VerticalAlignment = VerticalAlignment;
}
```

Note that you can also set the data item as the data context of the hyperlink instead of setting it in the **Source** property of the binding. For example:

- Visual Basic

```
Hyperlink.DataContext = row.DataItem
```

- C#

```
Hyperlink.DataContext = row.DataItem;
```

Although you will end up with the same result, this technique is does not perform as well as setting the binding source property directly.

Adding Properties to a Custom Column

You may want to add properties to a column in order to set a specific behavior. Continuing with the hyperlink column created in the previous topics, in this topic you'll add a property called **TargetName**. This property allows the user to specify the name of the target window or frame where the page will open.

Complete the following steps:

1. Add the following code to create the **TargetName** property:

   - Visual Basic

   ```
   Private _TargetName As String
   Public Property TargetName() As String
       Get
           Return _TargetName
       End Get
       Set(ByVal value As String)
           _TargetName = value
       End Set
   End Property
   ```

   - C#

   ```
   public string TargetName { get; set; }
   ```

2. Once the property is created you'll propagate this to the hyperlink in the **BindCellContent** method:

   - Visual Basic

   ```
   Public Overloads Overrides Sub BindCellContent(ByVal cellContent As 
   FrameworkElement, ByVal row As DataGridRow)
       Dim hyperlink = DirectCast(cellContent, HyperlinkButton)
       If Binding IsNot Nothing Then
           Dim newBinding As Binding = CopyBinding(Binding)
           newBinding.Source = row.DataItem
           hyperlink.SetBinding(HyperlinkButton.NavigateUriProperty, 
   newBinding)
       End If
       hyperlink.HorizontalAlignment = HorizontalAlignment
       hyperlink.VerticalAlignment = VerticalAlignment
       hyperlink.TargetName = TargetName
   End Sub
   ```

- C#
```
public override void BindCellContent(FrameworkElement cellContent,
DataGridRow row)
{
    var hyperlink = (HyperlinkButton)cellContent;
    if (Binding != null)
    {
        Binding newBinding = CopyBinding(Binding);
        newBinding.Source = row.DataItem;
        hyperlink.SetBinding(HyperlinkButton.NavigateUriProperty,
newBinding);
    }
    hyperlink.HorizontalAlignment = HorizontalAlignment;
    hyperlink.VerticalAlignment = VerticalAlignment;
    hyperlink.TargetName = TargetName;
}
```

Tips

You may find the following tips helpful when adding properties to a custom column:

- Provide a constructor that takes **PropertyInfo** as parameter calling base(property) in order to automatically set the **Binding**, **SortMemberPath**, **FilterMemberPath** and **Header** properties as well as properties set using custom attributes. Currently supported attributes include: **DisplayAttribute** (**AutoGenerateFilter**, **Name**, **GroupName**, **Order**), **DisplayFormatAttribute**, and **EditableAttribute**.

```
public DataGridHyperlinkColumn(PropertyInfo property) : base(property)
```

- You can set a converter in the binding to help you to manage scenarios where you need to use a column bound to property source that is not the same type. Suppose you want to bind a numeric column against a string property, this scenario will work correctly if you set a converter type which converts the string to a double.

Creating Custom Rows

You may be able to solve several scenarios by creating custom rows like a new row, group row, filter row, summary row, totals row, template row, and so on. Some of these rows are implemented internally and others are provided as samples.

When creating a custom row you'll be able to change the following parts:

- Cells content
- Row presenter
- Row header presenter

See Customizing Row Cell Content (page 67) for more details.

Customizing Row Cell Content

This topic explains how to customize cell content. For example, suppose you wanted to build a filter row. You could create a grid where the first row has a **TextBox** in each cell and when you type on it the grid is filtered by the typed text as in the following image:

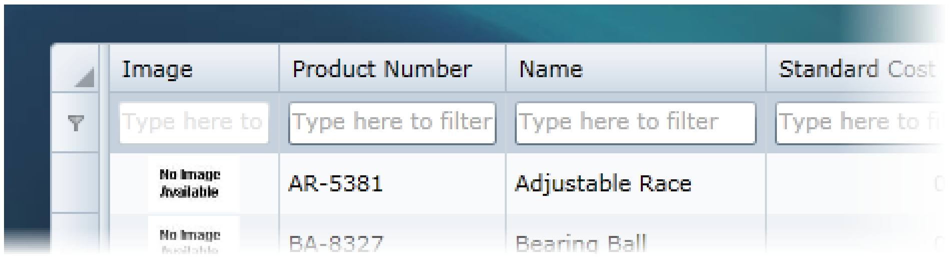


	Image	Product Number	Name	Standard Cost
	Type here to	Type here to filter	Type here to filter	Type here to fi
	No Image Available	AR-5381	Adjustable Race	
	No Image Available	BA-8327	Bearing Ball	

Adding a Class File

You would need to add a new class file where the custom row will be written. For example, complete the following steps to add a new class file:

1. Navigate to the Solution Explorer, right-click the project name and select **Add | New Item**.
2. In the **Add New Item** dialog box choose **Class** in the list of available templates.
3. Name the class, for example "DataGridFilterRow", and click the **Add** button to add the class to the project.
4. Update the class so it appears similar to the following:

   - Visual Basic

   ```
   Imports C1.Silverlight.DataGrid
   Public Class DataGridFilterRow
      Inherits DataGridRow
   End Class
   ```

   - C#

   ```
   using C1.Silverlight.DataGrid;
   public class DataGridFilterRow : DataGridRow
   {

   }
   ```

   This will update the class to inherit from **DataGridRow**. Once the file is created it must inherit from **DataGridRow**.

Once you've added the class, you can use it to implement filtering in the grid.

Overriding Methods

The methods you would need to override to specify the cell content of custom row are very similar to those exposed in custom columns. To implement custom cell content you'd need to override the following methods:

- **HasCellPresenter**: Determines whether a cell should exist for this row and the specified column.
- **GetCellContentRecyclingKey**: Key used to store the cell content for future reuse in a shared pool. Rows returning the same **RecyclingKey** can share the same cell content instances.
- **CreateCellContent**: Creates a visual element that will be used to display information inside a cell in this column.
- **BindCellContent**: Initializes the cell content presenter.
- **UnbindCellContent**: This method is called before the cell content is recycled.

In the filter row the **HasCellPresenter** method will return always true, because all columns will have a corresponding cell. In other scenarios like a summary row, only the columns where there is an aggregate function will have a cell.

The **GetCellContentRecyclingKey** method will return **typeof(TextBox)**, which allows recycling the text boxes, and the **CreateCellContent** will create a new instance of it. Add the following code to

- Visual Basic

```
Protected Overrides Function GetCellContentRecyclingKey(column As
DataGridColumn) As Object
      Return GetType(TextBox)
End Function

Protected Overrides Function CreateCellContent(column As DataGridColumn)
As FrameworkElement
      Return New TextBox()
End Function
```

- C#

```
protected override object GetCellContentRecyclingKey(DataGridColumn
column)
{
    return typeof(TextBox);
}

protected override FrameworkElement CreateCellContent(DataGridColumn
column)
{
    return new TextBox();
}
```

Implementing Filtering

In the previous steps you added a **TextBox** in each cell, but these controls currently do not do anything; to implement filtering complete the following steps:

1. Add the following code to the **BindCellContent** method:

   - Visual Basic

```
Protected Overrides Sub BindCellContent(cellContent As
FrameworkElement, column As DataGridColumn)
   Dim filterTextBox = DirectCast(cellContent, TextBox)
   'If the column doesn't have a FilterMemberPath specified
   'it won't allow entering text in the TextBox;
   If String.IsNullOrEmpty(column.FilterMemberPath) Then
         filterTextBox.IsEnabled = False
         filterTextBox.Text = "Not available"
   Else
         filterTextBox.Text = ""
         filterTextBox.IsEnabled = True
   End If
   ' Handle TextChanged to apply the filter to the column.
   filterTextBox.TextChanged += New EventHandler(Of
TextChangedEventArgs)(filterTextBox_TextChanged)
End Sub
```

   - C#

```
protected override void BindCellContent(FrameworkElement cellContent,
DataGridColumn column)
{
    var filterTextBox = (TextBox)cellContent;
    //If the column doesn't have a FilterMemberPath specified
    //it won't allow entering text in the TextBox;
    if (string.IsNullOrEmpty(column.FilterMemberPath))
    {
        filterTextBox.IsEnabled = false;
        filterTextBox.Text = "Not available";
    }
    else
    {
        filterTextBox.Text = "";
        filterTextBox.IsEnabled = true;
    }
    // Handle TextChanged to apply the filter to the column.
    filterTextBox.TextChanged += new
EventHandler<TextChangedEventArgs>(filterTextBox_TextChanged);
}
```

2. In **UnbindCellContent** you must remove the text changed handler to avoid leaking memory:

- Visual Basic

```
Protected Overrides Sub UnbindCellContent(cellContent As
FrameworkElement, column As DataGridColumn)
   Dim filterTextBox = DirectCast(cellContent, C1SearchBox)
   filterTextBox.TextChanged -= New EventHandler(Of
TextChangedEventArgs)(filterTextBox_TextChanged)
End Sub
```

- C#

```
protected override void UnbindCellContent(FrameworkElement cellContent,
DataGridColumn column)
{
    var filterTextBox = (C1SearchBox)cellContent;
    filterTextBox.TextChanged -= new
EventHandler<TextChangedEventArgs>(filterTextBox_TextChanged);
}
```

Adding a Custom Row to the Data Grid

You can replace rows the data grid uses to show the data of each data item or group with custom rows, or you can add custom rows on top or bottom of data item rows.

Replacing Data Item Row

In order to replace the rows generated by the data grid you must add a handler to the **CreatingRow** event. For example, in the following image the rows were replaced with template rows:

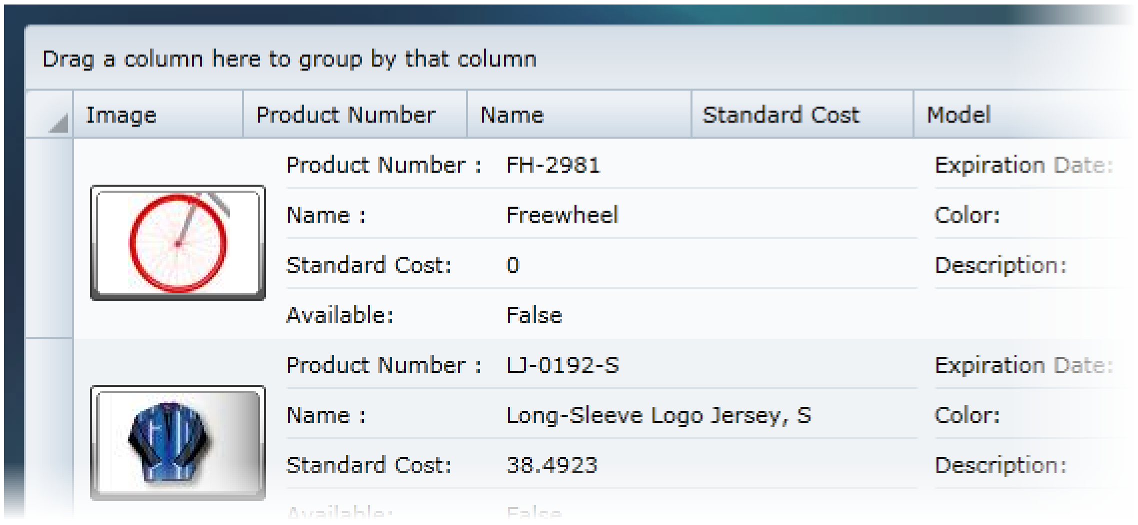

Drag a column here to group by that column

Image	Product Number	Name	Standard Cost	Model

Product Number : FH-2981 Expiration Date:
Name : Freewheel Color:
Standard Cost: 0 Description:
Available: False

Product Number : LJ-0192-S Expiration Date:
Name : Long-Sleeve Logo Jersey, S Color:
Standard Cost: 38.4923 Description:
Available: False

The following code replaces the default row with a template row:

- Visual Basic

```
Private Sub C1DataGrid_CreatingRow(sender As Object, e As
DataGridCreatingRowEventArgs)
      'Check if it's an item row (it could be a group row too).
      If e.Type = DataGridRowType.Item Then
            e.Row = New DataGridTemplateRow() With { _
                  .RowTemplate = DirectCast(Resources("TemplateRow"),
DataTemplate) _
            }
      End If
End Sub
```

- C#

```
private void C1DataGrid_CreatingRow(object sender,
DataGridCreatingRowEventArgs e)
{
    //Check if it's an item row (it could be a group row too).
    if (e.Type == DataGridRowType.Item)
    {
        e.Row = new DataGridTemplateRow()
        {
            RowTemplate = (DataTemplate)Resources["TemplateRow"]
        };
    }
   }
```

Adding an Extra Row

ComponentOne DataGrid for Silverlight allows adding one or more rows on top or bottom of data. This functionality is used in the new row, total row, summary row, and filter row scenarios.

For example, in XAML or code:

- XAML

```
<c1:C1DataGrid>
    <c1:C1DataGrid.TopRows>
        <local:DataGridFilterRow />
    </c1:C1DataGrid.TopRows>
    <c1:C1DataGrid.BottomRows>
        <local:DataGridFilterRow/>
    </c1:C1DataGrid.BottomRows>
</c1:C1DataGrid>
```

- Visual Basic
```
grid.Rows.TopRows.Add(New DataGridFilterRow())
```
- C#
```
   grid.Rows.TopRows.Add(new DataGridFilterRow());
```

Adding Row Details

Each grid row in **ComponentOne DataGrid for Silverlight** can be expanded to display a row details section. This row details section can display more details information about a specific row's content. The row details section is defined by a **DataTemplate**, **RowDetailsTemplate** that specifies the appearance of the section and the data to be displayed. For an example, see the RowDetailsTemplate (page 90) topic.

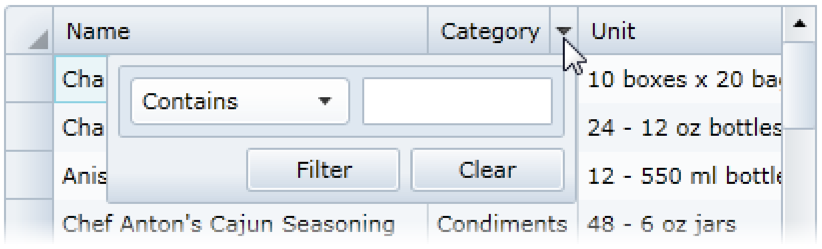
Using the **RowDetailsVisibilityMode** property the row details section can be displayed for selected rows, displayed for all rows, or it can be collapsed. See Setting Row Details Visibility (page 86) for more information.

Filtering the Grid

ComponentOne DataGrid for Silverlight includes several options for filtering the grid. You can add column filtering, a filter row, or full-text grid filtering. There's basic filtering or you can use the C1.Silverlight.DataGrid.Filters.dll assembly which offers more advanced filtering options than are built into the grid. How you choose to filter the grid will depends on your needs – for example if you just want the end user to be able to filter text in a column or if your application requires more advanced custom filtering.

Basic Column Filtering

For basic column filtering, simply set the CanUserFilter property to **True**. This will add a filter column element to the grid's user interface allowing end users to filter the grid via a drop-down box in each column's header:



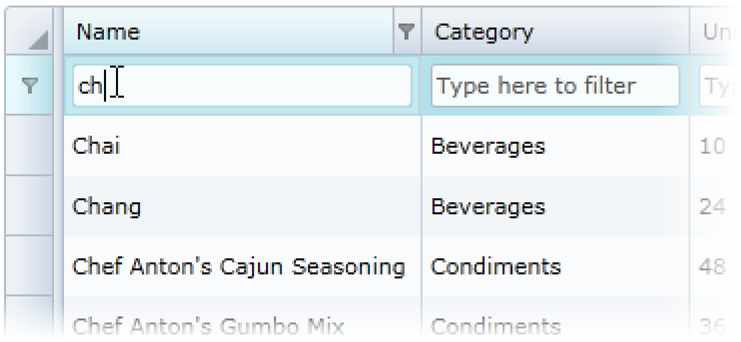
By default the CanUserFilter property will be set to **True** and filtering will be enabled. If you need to manually enable basic filtering, you can use the following markup or code:

- XAML
```
<c1:C1DataGrid Name="c1datagrid1" Height="180" Width="250"
CanUserFilter="True" />
```
- Visual Basic
```
Me.C1DataGrid1.CanUserFilter = True
```
- C#
```
this.c1DataGrid1.CanUserFilter = true;
```

See the Filtering Columns (page 95) topic for more details and examples.

Filter Row Filtering

If you choose, you can add a visible filter row column to the top or bottom of your grid. The filter row column appears as a row consisting of text boxes in each cell. When text is entered in a text box, the text of the column and grid is filtered by that text as it is entered:



Name	Category	Un
ch	Type here to filter	Ty
Chai	Beverages	10
Chang	Beverages	24
Chef Anton's Cajun Seasoning	Condiments	48
Chef Anton's Gumbo Mix	Condiments	36

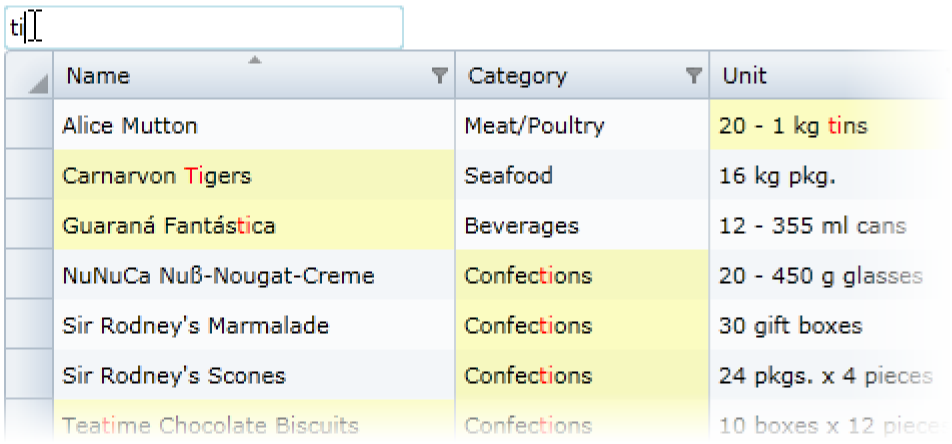
For example, the following markup adds two filter rows, one at the top and one at the bottom of the grid:

```
<c1:C1DataGrid x:Name="grid" Grid.Row="1" CanUserAddRows="False"
CanUserFreezeColumns="True" FrozenTopRowsCount="1" FrozenBottomRowsCount="1"
RowHeight="30" >

    <c1:C1DataGrid.TopRows>

        <c1:DataGridFilterRow />

    </c1:C1DataGrid.TopRows>

    <c1:C1DataGrid.BottomRows>

        <c1:DataGridFilterRow/>

    </c1:C1DataGrid.BottomRows>

</c1:C1DataGrid>
```

You can see the C1DataGrid_Demo2010/Filtering/FilterRow/FilterRow.xaml sample for an example.

Full Text Grid Filtering

C1DataGrid also supports full text filtering for the entire grid. Setting an attached property to the data grid allows the end user to filter the whole data grid (all the columns at once) by text entered in an external text box. All the matching results in the grid will be highlighted as the user types:



ti

Name	Category	Unit
Alice Mutton	Meat/Poultry	20 - 1 kg tins
Carnarvon Tigers	Seafood	16 kg pkg.
Guaraná Fantástica	Beverages	12 - 355 ml cans
NuNuCa Nuß-Nougat-Creme	Confections	20 - 450 g glasses
Sir Rodney's Marmalade	Confections	30 gift boxes
Sir Rodney's Scones	Confections	24 pkgs. x 4 pieces
Teatime Chocolate Biscuits	Confections	10 boxes x 12 piece

To use this method of grid filtering, you would need to add a text box control to your application and reference that control in the FullTextSearchBehavior attached property. For example, with the following XAML markup:

```
<StackPanel>
    <c1:C1TextBoxBase x:Name="filterTextBox" Width="200" Watermark = "Type
here to filter text"/>
    <c1:C1DataGrid x:Name="c1dg" c1:C1NagScreen.Nag="True">
    <c1:C1FullTextSearchBehavior.FullTextSearchBehavior>
        <c1:C1FullTextSearchBehavior Filter="{Binding
ElementName=filterTextBox,Path=C1Text}"/>
    </c1:C1FullTextSearchBehavior.FullTextSearchBehavior>
</c1:C1DataGrid>
</StackPanel>
```

You can see the C1DataGrid_Demo2010/Filtering/OneTextBoxFilter/OneTextBoxFilter.xaml sample for an example.

Advanced Filtering

One was to add advanced filtering is by using C1AdvancedFiltersBehavior. C1AdvancedFiltersBehavior adds a range of advanced filters to the C1DataGrid built-in columns. For example, this behavior adds several predefined filters, expanding the options for each column:

```
 <c1:C1DataGrid>
    <c1:C1AdvancedFiltersBehavior.AdvancedFiltersBehavior>
        <c1:C1AdvancedFiltersBehavior/>
    </c1:C1AdvancedFiltersBehavior.AdvancedFiltersBehavior>
</c1:C1DataGrid>
```

You can find a sample running in $\Silverlight\Main\Samples\SL\C1.Silverlight.DataGrid\C1DataGrid_Demo, file DemoGrid.xaml.

Column Filter List

An option for filtering the grid is to add a list of filters to a column in XAML. For example, the following markup adds three filters in a numeric column including a custom filter called RangeFilter:

```
<c1:DataGridNumericColumn Header="Range filter" Binding="{Binding
StandardCost}" FilterMemberPath="StandardCost">
    <c1:DataGridNumericColumn.Filter>
        <c1:DataGridContentFilter>
            <c1:DataGridFilterList>
                <local:DataGridRangeFilter Minimum="0" Maximum="1000"/>
                <c1:DataGridNumericFilter/>
                <c1:DataGridTextFilter/>
            </c1:DataGridFilterList>
        </c1:DataGridContentFilter>
    </c1:DataGridNumericColumn.Filter>
</c1:DataGridNumericColumn>
```

You can find a sample in $\Silverlight\Main\Samples\SL\C1.Silverlight.DataGrid\C1DataGrid_Demo, file Filtering/CustomFilters/CustomFilters.xaml.

Tab Filter List

An option for filtering the grid is to add a list of filters displayed in a tab control:

```
<c1:DataGridNumericColumn Header="Filters inside a tab control"
Binding="{Binding StandardCost}" FilterMemberPath="StandardCost">
    <c1:DataGridNumericColumn.Filter>
        <c1:DataGridContentFilter>
            <local:DataGridTabFilters Width="250">
                <local:DataGridRangeFilter Minimum="0" Maximum="1000"/>
                <c1:DataGridNumericFilter/>
                <c1:DataGridTextFilter/>
            </local:DataGridTabFilters>
        </c1:DataGridContentFilter>
    </c1:DataGridNumericColumn.Filter>
</c1:DataGridNumericColumn>
```

You can find a sample in $\Silverlight\Main\Samples\SL\C1.Silverlight.DataGrid\C1DataGrid_Demo, file Filtering/CustomFilters/CustomFilters.xaml.

Summarizing the Grid

ComponentOne DataGrid for Silverlight includes the C1.Silverlight.DataGrid.Summaries.dll assembly which can enhance the grid by adding a summary row.

The Summaries assembly contains the following features:

- SummaryRow. a row that shows the aggregate functions corresponding to each column (See C1DataGrid_Demo2010/Grouping/GrandTotal.xaml)
- GroupRowWithSummaries the same as the previous one but the summaries are shown in the group row rather than a regular row. (See C1DataGrid_Demo2010/Grouping/Grouping.xaml)

Localizing the Application

You can localize (translate) end user visible strings in **ComponentOne DataGrid for Silverlight**. Localization in **DataGrid for Silverlight** is based on the same approach as the standard localization of .NET WinForms applications.

To localize your application, you will need to complete the following steps:

1. Add resource files for each culture that you wish to support. See Adding Resource Files (page 76).
2. Update your project file's supported cultures. See Adding Supported Cultures (page 77).
3. And, depending on your project, set the current culture. See Setting the Current Culture (page 78).

The following topics describe localizing the grid in more detail.

Adding Resource Files

As with Windows Forms, you can create a set of resource files for the **DataGrid for Silverlight** assembly. You can create separate resource files, with the extension .resx, for each required culture. When the application runs you can switch between those resources and between languages. Note that all parts of your application using components from a **DataGrid for Silverlight** DLL must use the same localization resource.

Localization Conventions

To localize the grid you would need to set up resource files for each localized culture. The following conventions are recommended when creating .resx resource files:

- All .resx files should be placed in the **Resources** subfolder of your project.
- Files should be named as follows:

  XXX.YYY.resx, where:

  - XXX is the name of the ComponentOne assembly.
  - YYY is the culture code of the resource. If your translation is only for the invariant culture, the .resx file does not need to contain a culture suffix.

  For example:

  - C1.Silverlight.DataGrid.de.resx – German (de) resource for the C1.WPF.DataGrid assembly.
  - C1.Silverlight.DataGrid.resx – Invariant culture resource for the C1.WPF.DataGrid assembly.

Localization Strings

The following table lists strings that can be added to an .resx file to localize your application:

String	Default Value	Description
AddNewRow	Click here to add a new row	Text that appears in the add new row.
CheckBoxFilter_Checked	Checked :	Text that appears in the filter for check box columns to indicate if the column should be filtered for checked or unchecked items.
ComboBoxFilter_SelectAll	Select All	Text that appears in the filter for check box columns to select all items.
DateTimeFilter_End	End	Text that appears in the filter for date time columns for the end of the date time range.
DateTimeFilter_Start	Start	Text that appears in the filter for date time columns for the start of the date time range.
EmptyGroupPanel	Drag a column here to group	Text that appears in the grouping area of the

	by that column	grid when no columns are grouped.
Filter_Clear	Clear	Text that appears in the filter bar to clear the filter condition.
Filter_Filter	Filter	Text that appears in the filter bar to add a filter condition.
NumericFilter_And	And	Text that appears in the filter bar for numeric columns to indicate multiple filter conditions.
NumericFilter_Equals	Equals	Text that appears in the filter bar for numeric columns to indicate the filter condition should apply to exact matches only.
NumericFilter_GraterOrEquals	Greater/Equals	Text that appears in the filter bar for numeric columns to indicate the filter condition should apply to items with higher values than the condition value or exact matches only.
NumericFilter_Greater	Greater	Text that appears in the filter bar for numeric columns to indicate the filter condition should apply to items with higher values than the condition value.
NumericFilter_Less	Less	Text that appears in the filter bar for numeric columns to indicate the filter condition should apply to items with lower values than the condition value.
NumericFilter_LessOrEquals	Less/Equals	Text that appears in the filter bar for numeric columns to indicate the filter condition should apply to items with lower values than the condition value or exact matches only.
NumericFilter_NotEquals	Not Equals	Text that appears in the filter bar for numeric columns to indicate the filter condition should apply to items that are not an exact match.
NumericFilter_Or	Or	Text that appears in the filter bar for numeric columns to indicate multiple filter conditions.
TextFilter_Contains	Contains	Text that appears in the filter for text columns to indicate if the filter condition should apply to items that contain the value of the condition.
TextFilter_Equals	Equals	Text that appears in the filter bar for text columns to indicate the filter condition should apply to exact matches only.
TextFilter_NotEquals	Not Equals	Text that appears in the filter bar for text columns to indicate the filter condition should apply to items that are not an exact match.
TextFilter_StartsWith	Starts With	Text that appears in the filter for text columns to indicate if the filter condition should apply to items that start with the value of the condition.

Adding Supported Cultures

Once you've created resource files for your application, you will need to set the supported cultures for your project. To do so, complete the following steps:

1. In the Solution Explorer, right-click your project and select **Unload Project**.

   The project will appear grayed out and unavailable.

2. Right click the project again, and select the **Edit ProjectName.csproj** option (or **Edit ProjectName.vbproj**, where *ProjectName* is the name of your project).

3. In the .csproj file, locate the `<SupportedCultures></SupportedCultures>` tags. In between the tags, list the cultures you want to be supported, separating each with a semicolon.

   For example:
   ```
   <SupportedCultures>fr;es;en;it;ru</SupportedCultures>
   ```
   This will support French, Spanish, English, Italian, and Russian.

4. Save and close the .csproj or .vbproj file.

5. In the Solution Explorer, right-click your project and choose **Reload Project** from the content menu.

   The project will be reloaded and will now support the specified cultures.

Setting the Current Culture

The **C1DataGrid** control will use localization files automatically according to the culture selected in the application as long as you haven't moved files to another location or excluded files from the project. By default, the current culture is designated as **System.Threading.Thread.CurrentThread.CurrentUICulture**. If you want to use a culture other than the current culture, you can set the desired culture in your application using the following code:

- Visual Basic
```
Public Sub New()
      ' Set desired culture, for example here the French (France) locale.
      System.Threading.Thread.CurrentThread.CurrentUICulture = New
System.Globalization.CultureInfo("fr-FR")
      ' InitializeComponent() call.
      ' Add any initialization after the InitializeComponent() call.
      InitializeComponent()
End Sub
```

- C#
```
public MainPage()
{
    // Set desired culture, for example here the French (France) locale.
    System.Threading.Thread.CurrentThread.CurrentUICulture = new
System.Globalization.CultureInfo("fr-FR");
    // InitializeComponent() call.
    InitializeComponent();
    // Add any initialization after the InitializeComponent() call.
}
```

Enabling or Disabling End User Interaction

You can customize how much control end users have over the grid at run time. For example you can enable grouping, and prevent actions such as filtering columns and resizing rows. The following table lists properties that you can use to customize how users interact with the grid:

Property	**Description**
CanUserAddRows	Determines if users can add rows at run time. **True** by default.
CanUserEditRows	Determines if users can edit rows at run time. **True** by default.
CanUserFilter	Determines if users can filter columns at run time. If **True**, the filter bar will be visible on columns. **True** by default.
CanUserGroup	Determines if users can group rows at run time. If **True** the grouping area of the grid will be visible. **False** by default.
CanUserRemoveRows	Determines if users can remove rows at run time by pressing

	the DELETE key. **True** by default.
CanUserReorderColumns	Determines if users can reorder columns at run time by using a drag-and-drop operation. **True** by default.
CanUserResizeColumns	Determines if users can resize columns at run time. **True** by default.
CanUserResizeRows	Determines if users can resize rows at run time. **False** by default.
CanUserSort	Determines if users can sort columns at run time by clicking on a column's header. **True** by default.
CanUserToggleDetails	Determines if users can toggle the row details section's visibility. **True** by default.
CanUserFreezeColumns	Determines if users can change the number of frozen columns by dragging the freezing separator at run time. **None** by default.

In each column you can customize the following properties:

Property	**Description**
CanUserMove	Determines if users can reorder this column at run time. **True** by default.
CanUserResize	Determines if users can resize this column at run time. **True** by default.
CanUserFilter	Determines if users can filter this column at run time. If **True**, the filter bar will be visible on this column. **True** by default.
CanUserSort	Determines if users can sort this column at run time. **True** by default.

Note: The properties set in the grid take precedence over those set in columns.

Setting Selection Mode

You can set the grid's selection mode behavior by setting the **SelectionMode** property. You can change how users interact with the grid, but setting the **SelectionMode** property to one of the following values:

Option	**Description**
None	The user cannot select any item.
SingleCell	The user can select only one cell at a time.
SingleRow	The user can select only one row at a time.
SingleColumn	The user can select only one column at a time.
SingleRange	The user can select only one cells range at a time. (A range is the rectangle delimited by two cells)
MultiRow (Default)	The user can select multiple rows while holding down the corresponding modifier key.
MultiColumn	The user can select multiple columns while holding down the corresponding modifier key.
MultiRange	The user can select multiple cells ranges while holding down the corresponding modifier key.

For more information about modifier keys and the **MultiRow** option, see the Multiple Row Selection (page 93) topic.

Locking the Grid

By default users can interact and edit the grid and columns in the grid. If you choose, you can set the grid or specific columns in the grid to not be editable with the **IsReadOnly** property.

In XAML

To lock the grid from being edited, add `IsReadOnly="True"` to the `<c1:C1DataGrid>` tag so that it appears similar to the following:

```
<c1:C1DataGrid x:Name="C1DataGrid1" IsReadOnly="True">
```

In Code

To lock the grid from editing, set the **IsReadOnly** property to **True**. For example:

- Visual Basic
  ```
  Me.C1DataGrid1.IsReadOnly = True
  ```
- C#
  ```
  this.c1DataGrid1.IsReadOnly = true;
  ```

Deferred and Real Time Scrolling

ComponentOne DataGrid for Silverlight supports both real time and deferred scrolling. By default, real time scrolling is used and as a user moves the thumb button or clicks the scroll button the grid scrolls. In deferred scrolling, the grid is not scrolled until the user releases the scrollbar thumb; the grid does not move as the scrollbar thumb is moved. You might want to implement deferred scrolling in your application if the grid contains a large amount of data or to optimize scrolling.

You can determine how the grid is scrolled by setting the ScrollMode property. You can set the ScrollMode property to a C1DataGridScrollMode enumeration option, either **RealTime** (default) or **Deferred**. The example below set the grid to deferred scrolling mode.

In XAML

To set the grid to deferred scrolling mode, add `ScrollMode="Deferred"` to the `<c1:C1DataGrid>` tag so that it appears similar to the following:

```
<c1:C1DataGrid x:Name="c1DataGrid1" ScrollMode="Deferred">
```

In Code

To set the grid to deferred scrolling mode, set the ScrollMode property to **Deferred**. For example:

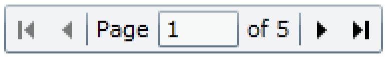
- Visual Basic
  ```
  Me.C1DataGrid1.ScrollMode = C1DataGridScrollMode.Deferred
  ```
- C#
  ```
  this.c1DataGrid1.ScrollMode = C1DataGridScrollMode.Deferred;
  ```

Paging Grid Data

If you are displaying a large amount of data in the grid or have a limited amount of space in your application, you might want to add paging to your **C1DataGrid** control. **ComponentOne DataGrid for Silverlight** supports paging through the use of the C1DataPager control and the **PagedCollectionView** class. Paging the grid can decrease load time and allow users to interact more easily with the control.

C1DataPager Control

The C1DataPager control is very similar to the standard Microsoft **DataPager** control. When you add the control to your application, it will appear similar to the following image:



The control includes **First**, **Previous**, **Next**, and **Last** buttons by default as well as a text box listing the current page and total number of pages. You can include this control in your application, and by setting the **PageSize** property on the control allow the grid to be paged by any amount you choose.

PagedCollectionView Class

The C1DataPager control provides a convenient user interface for controlling paging with a **PagedCollectionView**. You use the **PagedCollectionView** class to provide grouping, sorting, filtering, and paging functionality for any collection that implements the **IEnumerable** interface. You can think of a collection view as a layer on top of a binding source collection that allows you to navigate and display the collection based on sort, filter, and group queries, all without having to manipulate the underlying source collection itself.

Using the C1DataPager Control and PagedCollectionView Class

So suppose you might set the C1DataGrid control's ItemsSource property with the following code:

- Visual Basic
```
C1DataGrid1.ItemsSource = data
```
- C#
```
c1DataGrid1.ItemsSource = data;
```

Instead, using the **PagedCollectionView** class, you might set the C1DataGrid control's ItemsSource property with the following code:

- Visual Basic
```
C1DataGrid1.ItemsSource = data
```
- C#
```
c1DataGrid1.ItemsSource = new PagedCollectionView(data);
```

And then you might bind the C1DataPager control to the C1DataGrid control; for example in XAML markup:

```
<c1ria:C1DataPager Source="{Binding ItemsSource, ElementName=c1DataGrid1}"
HorizontalAlignment="Left" Name="c1DataPager1" VerticalAlignment="Top"
PageSize="5" />
```

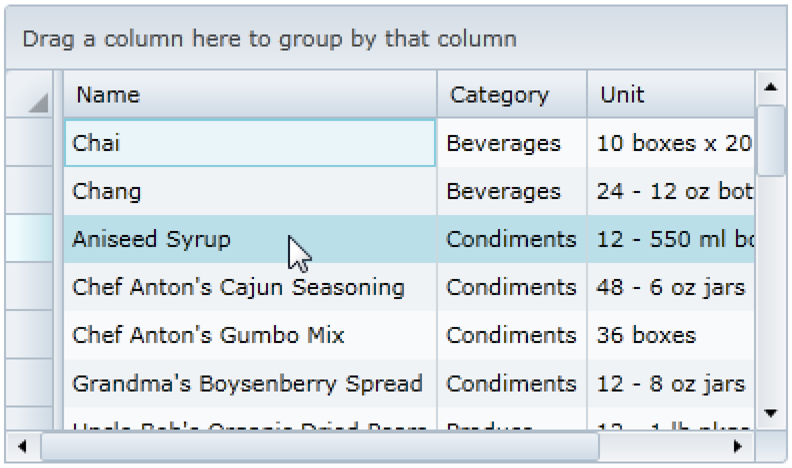
DataGrid for Silverlight's Appearance

The **C1DataGrid** control supports common table formatting options, such as alternating row backgrounds and the ability to show or hide headers, grid lines, and scroll bars. Additionally, the control provides several brush, style and template properties that you can use to completely change the appearance of the control and its rows, columns, headers, and cells.

Note that **ComponentOne DataGrid for Silverlight** uses ClearStyle technology for styling. For details, see C1DataGrid ClearStyle (page 87).

C1DataGrid Themes

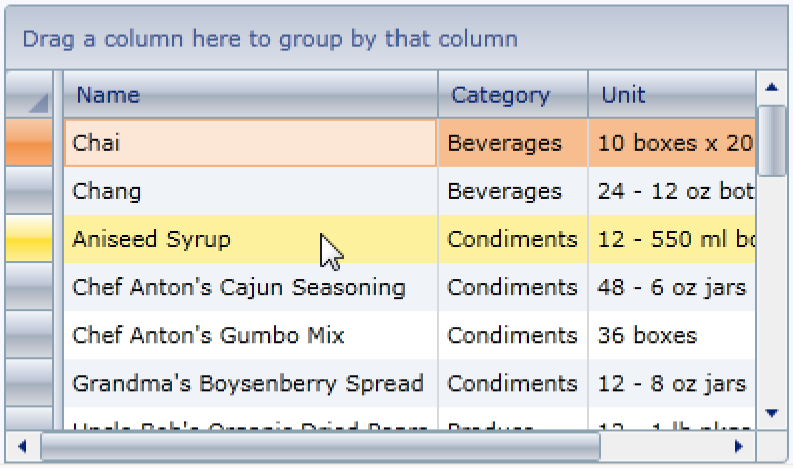
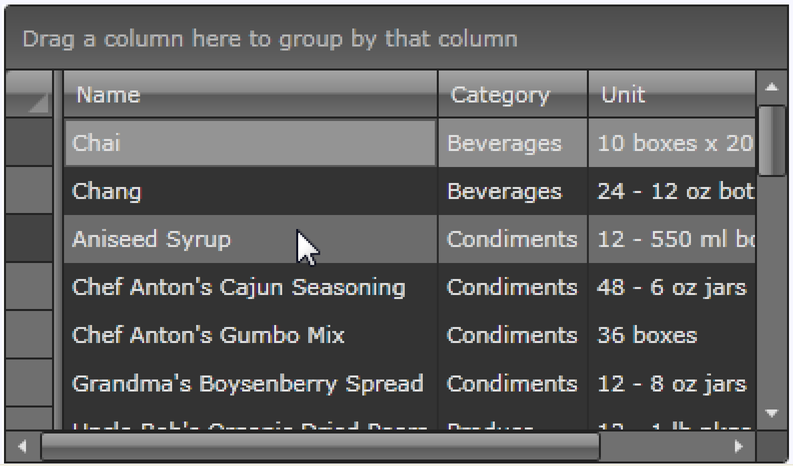
ComponentOne DataGrid for Silverlight incorporates several themes that allow you to customize the appearance of your grid. When you first add a **C1DataGrid** control to the page, it appears similar to the following image:



Drag a column here to group by that column

Name	Category	Unit
Chai	Beverages	10 boxes x 20
Chang	Beverages	24 - 12 oz bot
Aniseed Syrup	Condiments	12 - 550 ml bo
Chef Anton's Cajun Seasoning	Condiments	48 - 6 oz jars
Chef Anton's Gumbo Mix	Condiments	36 boxes
Grandma's Boysenberry Spread	Condiments	12 - 8 oz jars

This is the control's default appearance. You can change this appearance by using one of the built-in themes or by creating your own custom theme. All of the built-in themes are based on Silverlight Toolkit themes. The built-in themes are described and pictured below; note that in the images below, a cell has been selected and the mouse is hovering over another cell to show both selected and hover styles:

Theme Name	Theme Preview
C1ThemeBureauBlack	
C1ThemeExpressionDark	

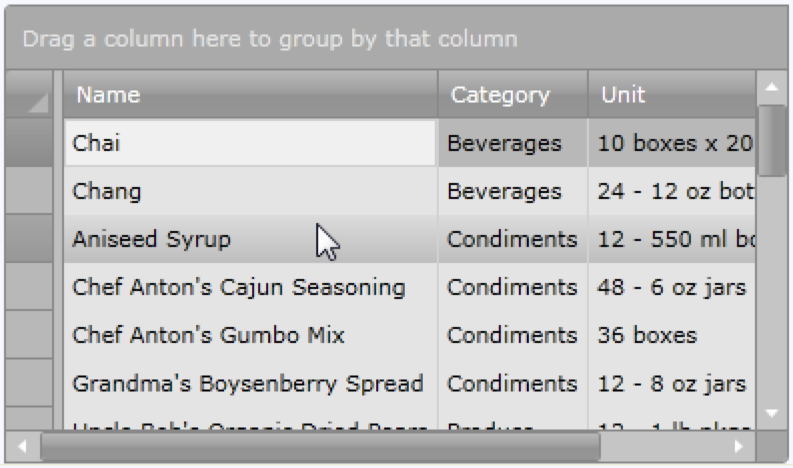
Drag a column here to group by that column

Name	Category	Unit
Chai	Beverages	10 boxes x 20
Chang	Beverages	24 - 12 oz bot
Aniseed Syrup	Condiments	12 - 550 ml bo
Chef Anton's Cajun Seasoning	Condiments	48 - 6 oz jars
Chef Anton's Gumbo Mix	Condiments	36 boxes
Grandma's Boysenberry Spread	Condiments	12 - 8 oz jars

Drag a column here to group by that column

Name	Category	Unit
Chai	Beverages	10 boxes x 20
Chang	Beverages	24 - 12 oz bot
Aniseed Syrup	Condiments	12 - 550 ml bo
Chef Anton's Cajun Seasoning	Condiments	48 - 6 oz jars
Chef Anton's Gumbo Mix	Condiments	36 boxes
Grandma's Boysenberry Spread	Condiments	12 - 8 oz jars

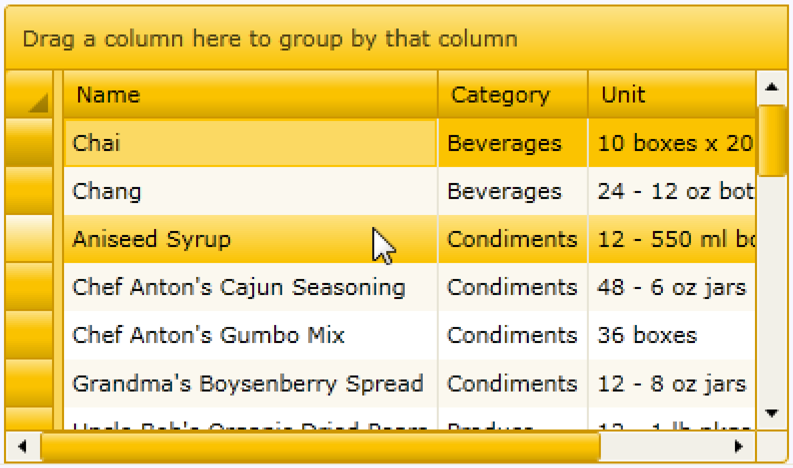
C1ThemeExpressionLight



Drag a column here to group by that column

Name	Category	Unit
Chai	Beverages	10 boxes x 20
Chang	Beverages	24 - 12 oz bot
Aniseed Syrup	Condiments	12 - 550 ml bo
Chef Anton's Cajun Seasoning	Condiments	48 - 6 oz jars
Chef Anton's Gumbo Mix	Condiments	36 boxes
Grandma's Boysenberry Spread	Condiments	12 - 8 oz jars

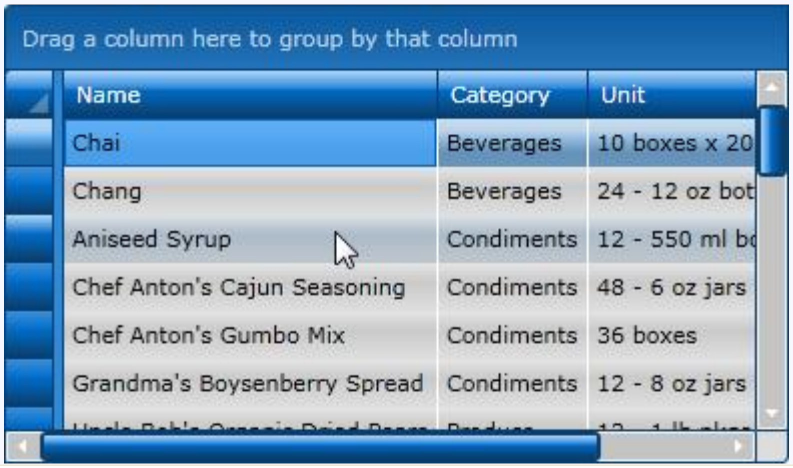
C1ThemeRainierOrange



Drag a column here to group by that column

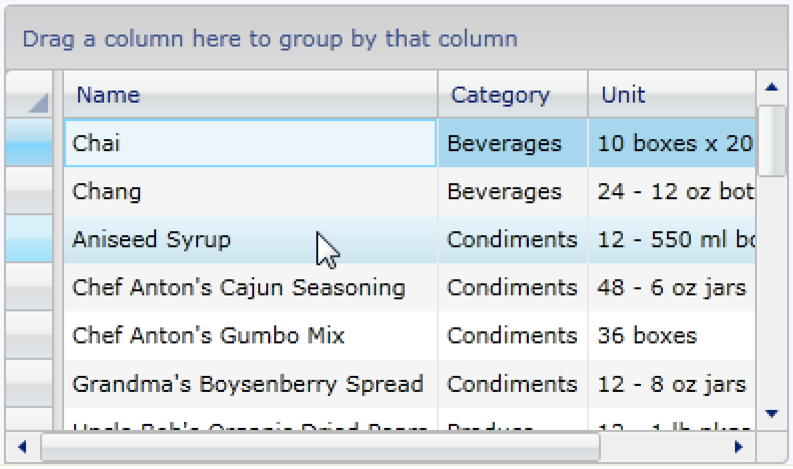
Name	Category	Unit
Chai	Beverages	10 boxes x 20
Chang	Beverages	24 - 12 oz bot
Aniseed Syrup	Condiments	12 - 550 ml bo
Chef Anton's Cajun Seasoning	Condiments	48 - 6 oz jars
Chef Anton's Gumbo Mix	Condiments	36 boxes
Grandma's Boysenberry Spread	Condiments	12 - 8 oz jars

C1ThemeShinyBlue



Drag a column here to group by that column

Name	Category	Unit
Chai	Beverages	10 boxes x 20
Chang	Beverages	24 - 12 oz bot
Aniseed Syrup	Condiments	12 - 550 ml bo
Chef Anton's Cajun Seasoning	Condiments	48 - 6 oz jars
Chef Anton's Gumbo Mix	Condiments	36 boxes
Grandma's Boysenberry Spread	Condiments	12 - 8 oz jars

C1ThemeWhistlerBlue	

Drag a column here to group by that column

Name	Category	Unit
Chai	Beverages	10 boxes x 20
Chang	Beverages	24 - 12 oz bot
Aniseed Syrup	Condiments	12 - 550 ml bo
Chef Anton's Cajun Seasoning	Condiments	48 - 6 oz jars
Chef Anton's Gumbo Mix	Condiments	36 boxes
Grandma's Boysenberry Spread	Condiments	12 - 8 oz jars

Editing Styles

ComponentOne DataGrid for Silverlight's **C1DataGrid** control provides several style properties that you can use to completely change the appearance of the control and its rows, columns, headers, and cells. Some of the included styles are described in the table below:

Style	Description
CellStyle	Gets or sets the style that is used when rendering the cells.
ColumnHeaderStyle	Gets or sets the style that is used when rendering the column headers.
DragOverColumnStyle	Style applied to a **ContentControl** element used to show the dragged column while it is moved.
DragSourceColumnStyle	Style applied to a **ContentControl** that is placed over the source column when it starts the drag-and-drop operation.
DropIndicatorStyle	Style applied to a **ContentControl** element used to indicate the position where the dragged column will be dropped.
FilterStyle	Gets or sets the style used for the filter control container.
FocusStyle	Sets the style of the internal **Rectangle** used to show the focus on the **C1DataGrid**.
GroupColumnHeaderStyle	Gets or sets the style that is used when rendering the column headers in the group panel.
GroupRowHeaderStyle	Gets of sets the style of the header of the group row.
GroupRowStyle	Gets of sets the style of the group row.
NewRowHeaderStyle	Gets or sets the style that is used when rendering the row header for entering new items.
NewRowStyle	Gets or sets the style that is used when rendering the row for entering new items.
RowHeaderStyle	Gets or sets the style that is used when rendering the row headers.
RowStyle	Gets or sets the style that is used when rendering the rows.

Table Formatting Options

The following topics detail table formatting options, including grid headers and placement of table objects.

Setting Row and Column Header Visibility

By default row and column headers are visible in the grid. However, if you choose, you can set one or both of the headers to be hidden by setting the **HeadersVisibility** property. You can set the **HeadersVisibility** property to one of the following options:

Option	Description
None	Neither row nor column headers are visible in the grid.
Column	Only column headers are visible in the grid.
Row	Only row headers are visible in the grid.
All (default)	Both column and row headers are visible in the grid.

Setting Grid Line Visibility

By default vertical and horizontal grid lines are visible in the grid. However, if you choose, you can set one or both sets of grid lines to be hidden by setting the **GridLinesVisibility** property. You can set the **GridLinesVisibility** property to one of the following options:

Option	Description
None	Neither horizontal nor vertical grid lines are visible in the grid.
Horizontal	Only horizontal grid lines are visible in the grid.
Vertical	Only vertical grid lines are visible in the grid.
All (default)	Both horizontal and vertical grid lines are visible in the grid.

Setting New Row Visibility

By default the Add New row is located at the bottom of the grid. However, if you choose, you can change its location by setting the **NewRowVisibility** property. You can set the **NewRowVisibility** property to one of the following options:

Option	Description
Top	The Add New row appears at the top of the grid.
Bottom (default)	The Add New row appears at the bottom of the grid.

Setting Vertical and Horizontal Scrollbar Visibility

By default the grid's horizontal and vertical scrollbars are only visible when the height or width of grid content exceeds the size of the grid. However, if you choose, you can set the scrollbars to be always or never visible, and even disable them altogether, by setting the **VerticalScrollbarVisibility** and **HorizontalScrollbarVisibility** properties. You can set the **VerticalScrollbarVisibility** and **HorizontalScrollbarVisibility** properties to one of the following options:

Option	Description
Disabled	The chosen scrollbar is disabled.
Auto (default)	The chosen scrollbar appears only when the content of the grid is exceeds the grid window.
Hidden	The chosen scrollbar appears to be hidden.
Visible	The chosen scrollbar is always visible.

Setting Row Details Visibility

By default row details are collapsed and not visible. You can use the **RowDetailsVisibilityMode** property to set if and when row details are visible. You can set the **RowDetailsVisibilityMode** property to one of the following options:

Option	Description
VisibleWhenSelected	Row details are only visible when selected.
Visible	Row details are always visible.
Collapsed (default)	Row details appear collapsed and are not visible.

C1DataGrid Brushes

ComponentOne DataGrid for Silverlight's **C1DataGrid** control provides several brush properties that you can use to completely change the appearance of the control and its rows, columns, headers, and cells. Some of the included brushes are described in the table below:

Brush	Description
Background	Gets or sets the background brush that is used when rendering. (This brush will be applied to all the parts of the data grid)
Foreground	Gets or sets the foreground brush that is used when rendering. (This brush will be applied to all the parts of the data grid)
BorderBrush	Gets or sets the border brush that is used when rendering. (This brush will be applied to some of the parts of the data grid depending on the theme)
SelectedBrush	Gets or sets the selected brush that is used when rendering selected rows and row and column headers, etc.
MouseOverBrush	Gets or sets the mouse over brush that is used when mouse is over rows and row and column headers, etc.
RowBackground	Gets or sets the background brush of a row.
RowForeground	Gets or sets the foreground brush of a row.
AlternatingRowBackground	Gets or sets the background brush of an alternating row.
AlternatingRowForeground	Gets or sets the foreground brush of an alternating row.
HorizontalGridLinesBrush	Gets of sets the brush applied to the horizontal lines.
VerticalGridLinesBrush	Gets of sets the brush applied to the vertical lines.

ComponentOne DataGrid for Silverlight uses ClearStyle technology for styling. For details, see C1DataGrid ClearStyle (page 87).

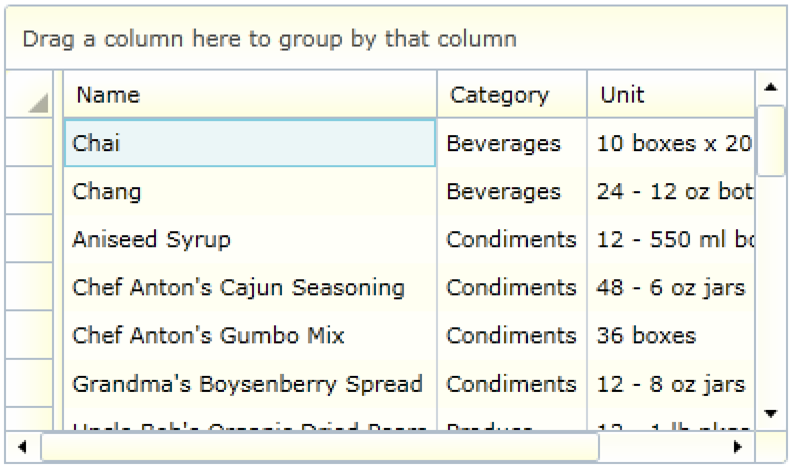
C1DataGrid ClearStyle

DataGrid for Silverlight supports ComponentOne's new ClearStyle technology that allows you to easily change control colors without having to change control templates. By just setting a few color properties you can quickly style the entire grid.

You can completely change the appearance of the **C1DataGrid** control by simply setting a few properties, such as the **C1DataGrid.Background** property which sets the color scheme of the **C1DataGrid** control. For example, if you set the **Background** property to "#FF663366" so the XAML markup appears similar to the following:

```
<c1:C1DataGrid HorizontalAlignment="Left" Margin="10,10,0,0"
Name="c1DataGrid1" VerticalAlignment="Top" CanUserFreezeColumns="Left"
CanUserGroup="True" Background="#FFFFFFCC"/>
```

The grid will appear similar to the following image:



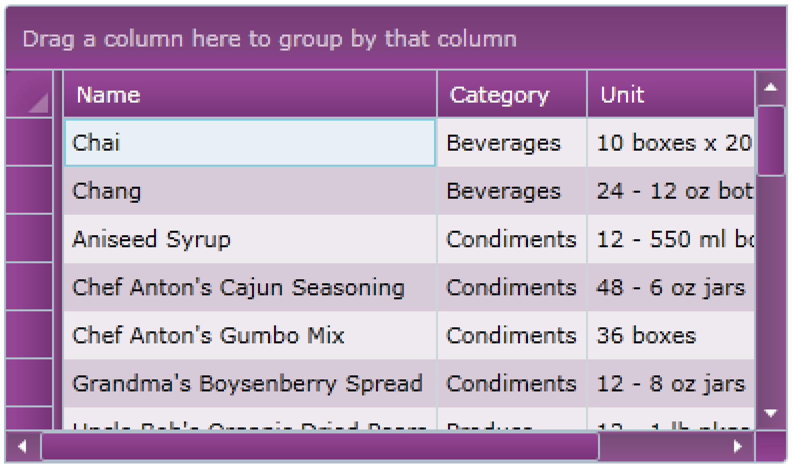
Drag a column here to group by that column

Name	Category	Unit
Chai	Beverages	10 boxes x 20
Chang	Beverages	24 - 12 oz bot
Aniseed Syrup	Condiments	12 - 550 ml b
Chef Anton's Cajun Seasoning	Condiments	48 - 6 oz jars
Chef Anton's Gumbo Mix	Condiments	36 boxes
Grandma's Boysenberry Spread	Condiments	12 - 8 oz jars

If you set the **Background** property to "#FF663366" and the **Foreground** property to "White", so the XAML markup appears similar to the following:

```
<c1:C1DataGrid HorizontalAlignment="Left" Margin="10,10,0,0"
Name="c1DataGrid1" VerticalAlignment="Top" CanUserFreezeColumns="Left"
CanUserGroup="True" Background="#FF663366" Foreground="White"/>
```

The grid will appear similar to the following image:



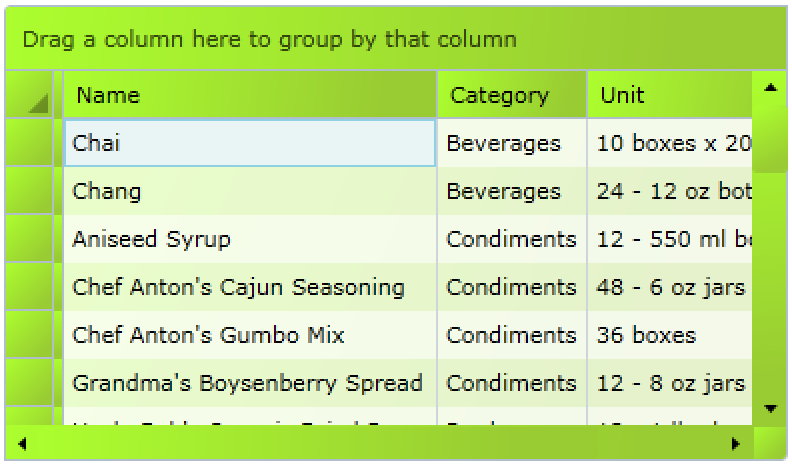
Drag a column here to group by that column

Name	Category	Unit
Chai	Beverages	10 boxes x 20
Chang	Beverages	24 - 12 oz bot
Aniseed Syrup	Condiments	12 - 550 ml b
Chef Anton's Cajun Seasoning	Condiments	48 - 6 oz jars
Chef Anton's Gumbo Mix	Condiments	36 boxes
Grandma's Boysenberry Spread	Condiments	12 - 8 oz jars

You can even set the **Background** property to a gradient value, for example with the following XAML:

```
<c1:C1DataGrid x:Name="c1DataGrid1" HorizontalAlignment="Left"
Margin="10,10,0,0" VerticalAlignment="Top" CanUserFreezeColumns="Left"
CanUserGroup="True">
    <c1:C1DataGrid.Background>
        <LinearGradientBrush StartPoint="0,0" EndPoint="1,1">
            <GradientStop Color="GreenYellow" Offset="0.0" />
            <GradientStop Color="YellowGreen" Offset="0.85" />
        </LinearGradientBrush>
    </c1:C1DataGrid.Background>
</c1:C1DataGrid>
```

The grid will appear similar to the following image:

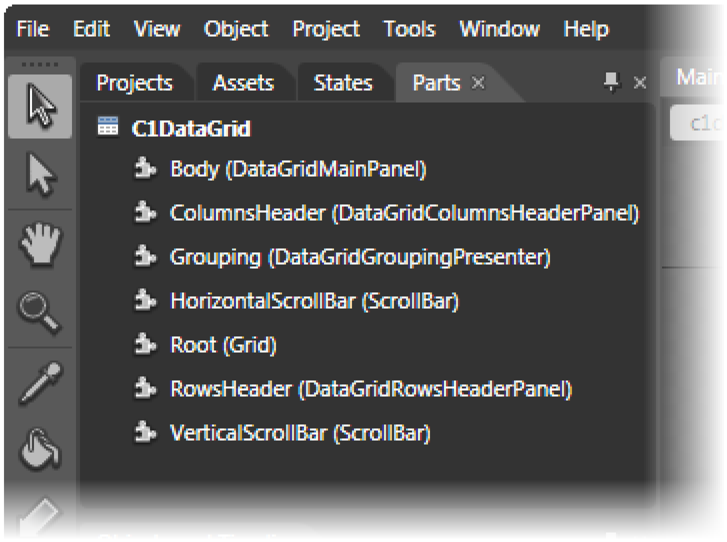


Drag a column here to group by that column

Name	Category	Unit
Chai	Beverages	10 boxes x 20
Chang	Beverages	24 - 12 oz bot
Aniseed Syrup	Condiments	12 - 550 ml b
Chef Anton's Cajun Seasoning	Condiments	48 - 6 oz jars
Chef Anton's Gumbo Mix	Condiments	36 boxes
Grandma's Boysenberry Spread	Condiments	12 - 8 oz jars

C1DataGrid Template Parts

In Microsoft Expression Blend, you can view and edit template parts by creating a new template (for example, click the **C1DataGrid** control to select it and choose **Object | Edit Template | Edit a Copy**). Once you've created a new template, the parts of the template will appear in the **Parts** window:



C1DataGrid
Body (DataGridMainPanel)
ColumnsHeader (DataGridColumnsHeaderPanel)
Grouping (DataGridGroupingPresenter)
HorizontalScrollBar (ScrollBar)
Root (Grid)
RowsHeader (DataGridRowsHeaderPanel)
VerticalScrollBar (ScrollBar)

Note that you may have to select the **ControlTemplate** for its parts to be visible in the **Parts** window.

In the Parts window, you can double-click any element to create that part in the template. Once you have done so, the part will appear in the template and the element's icon in the **Parts** pane will change to indicate selection:

Template parts available in the **C1DataGrid** control include:

Name	**Type**	**Description**
Body	**DataGridMainPanel**	Panel that contains the body of the grid.
ColumnsHeader	DataGridColumnsHeaderPanel	Panel that contains a collection of DataGridColumnsHeaderPanel.
Grouping	DataGridGroupingPresenter	Presenter that displays the grouping panel or another element if there is no columns in the grouping panel.
HorizontalScrollBar	ScrollBar	Represents a control that provides a scroll bar that has a sliding Thumb whose position corresponds to a value.
Root	Grid	Defines a flexible grid area that consists of columns and rows.
RowsHeader	DataGridRowsHeaderPanel	Panel that contains DataGridRowsHeaderPanel.
VerticalScrollBar	ScrollBar	Represents a control that provides a scroll bar that has a sliding Thumb whose position corresponds to a value.

RowDetailsTemplate

The **RowDetailsTemplate** template controls the appearance of the row details area. The row details section appears below a row and can display additional information.

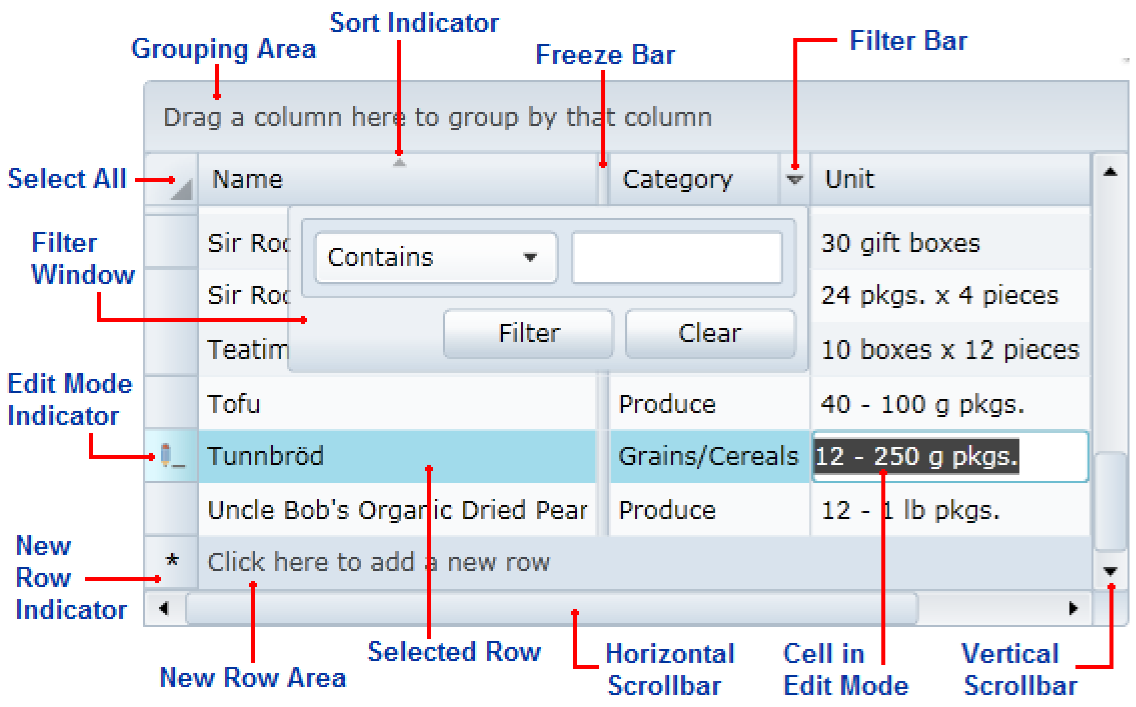
In Expression Blend, you can create an empty template at design time by selecting the **C1DataGrid** control and then clicking **Object | Edit Other Templates | Edit RowDetailsTemplate | Create Empty**.

You can include text, controls, and more in the **RowDetailsTemplate**, including controls bound to data. For example, the following template includes bound and unbound text and check boxes:

```
<c1:C1DataGrid.RowDetailsTemplate>
    <!-- Begin row details section. -->
    <DataTemplate>
        <Border BorderBrush="DarkGray" BorderThickness="1"
Background="Azure">
        <StackPanel Orientation="Horizontal">
            <StackPanel>
                <StackPanel Orientation="Horizontal">
                <!-- Controls are bound to properties. -->
                <TextBlock FontSize="16" Foreground="MidnightBlue"
Text="{Binding Name}" Margin="0,0,10,0" VerticalAlignment="Bottom" />
                <TextBlock FontSize="12" Text="Order Date: "
VerticalAlignment="Bottom"/>
                <TextBlock FontSize="12" Text=" Complete:"
VerticalAlignment="Bottom" />
                <CheckBox IsChecked="{Binding Complete, Mode=TwoWay}"
VerticalAlignment="Center" />
                </StackPanel>
                <TextBlock FontSize="12" Text="Notes: " />
                <TextBox FontSize="12" Text="{Binding Notes,
Mode=TwoWay}" Width="420" TextWrapping="Wrap"/>
            </StackPanel>
        </StackPanel>
        </Border>
    </DataTemplate>
    <!-- End row details section. -->
</c1:C1DataGrid.RowDetailsTemplate>
```

Run-time Interaction

The image below highlights some of the run-time interactions possible in a **ComponentOne DataGrid for Silverlight C1DataGrid** control:



The following topics detail these run-time features including filtering, sorting, and grouping data.

Keyboard and Mouse Navigation

ComponentOne DataGrid for WPF supports several run-time keyboard and mouse navigation options that provide increased accessibility. The following topics detail some of these end-user interactions.

Keyboard Navigation

The following table lists several keyboard shortcuts that can be used to navigate and manipulate the grid at run time. Note that on Apple computers, end users should use the Command (or Apple) key in place of the CTRL key:

Key Combination	Description
DOWN Arrow	Moves the focus to the cell directly below the current cell. If the focus is in the last row, pressing the DOWN ARROW does nothing.
UP Arrow	Moves the focus to the cell directly above the current cell. If the focus is in the first row, pressing the UP ARROW does nothing.
LEFT ARROW	Moves the focus to the previous cell in the row. If the focus is in the first cell in the row, pressing the LEFT ARROW does nothing.
RIGHT Arrow	Moves the focus to the next cell in the row. If the focus is in the last cell in the row, pressing the RIGHT ARROW does nothing.
HOME	Moves the focus to the first cell in the current row.
END	Moves the focus to the last cell in the current row.
PAGE DOWN	Scrolls the control downward by the number of rows that are displayed. Moves the focus to the last displayed row without changing columns. If the last row is only partially displayed, scrolls the grid to fully display the last row.
PAGE UP	Scrolls the control upward by the number of rows that are displayed.

	Moves focus to the first displayed row without changing columns. If the first row is only partially displayed, scrolls the grid to fully display the first row.
TAB	If the current cell is in edit mode, moves the focus to the next editable cell in the current row. If the focus is already in the last cell of the row, commits any changes that were made and moves the focus to the first editable cell in the next row. If the focus is in the last cell in the control, moves the focus to the next control in the tab order of the parent container. If the current cell is not in edit mode, moves the focus to the next control in the tab order of the parent container.
SHIFT+TAB	If the current cell is in edit mode, moves the focus to the previous editable cell in the current row. If the focus is already in the first cell of the row, commits any changes that were made and moves the focus to the last cell in the previous row. If the focus is in the first cell in the control, moves the focus to the previous control in the tab order of the parent container. If the current cell is not in edit mode, moves the focus to the previous control in the tab order of the parent container.
CTRL+DOWN ARROW	Moves the focus to the last cell in the current column.
CTRL+UP ARROW	Moves the focus to the first cell in the current column.
CTRL+RIGHT ARROW	Moves the focus to the last cell in the current row.
CTRL+LEFT ARROW	Moves the focus to the first cell in the current row.
CTRL+HOME	Moves the focus to the first cell in the control.
CTRL+PAGE DOWN	Same as PAGE DOWN.
CTRL+PAGE UP	Same as PAGE UP.
ENTER	Enter/exit edit mode on a selected cell (if the grid and column's **IsReadOnly** properties are **False**).
F2	Enter edit mode on a selected cell (if the grid and column's **IsReadOnly** properties are **False**). If the focus is on the new row, the grid begins editing the first editable cell of the new row.
ESC	Cancel editing of a cell or new row.
DEL	Delete selected row.
INSERT	Scrolls to the new row and begins editing it.

Mouse Navigation

The following table lists several mouse and keyboard shortcuts that can be used to navigate and manipulate the grid at run time. Note that on Apple computers, end users should use the Command (or Apple) key in place of the CTRL key:

Mouse Action	Description
Click an unselected row	Makes the clicked row the current row.
Click a cell in the current row	Puts the clicked cell into edit mode.
Drag a column header cell	Moves the column so that it can be dropped into a new position (if the **CanUserReorderColumns** property is **True** and the current column's **CanUserReorder** property is **True**).
Drag a column header separator	Resizes the column (if the **CanUserResizeColumns** property is **True** and the **CanUserResize** property is True for the current column).

Click a column header cell	If the property **ColumnHeaderClickAction** is set to **Sort**, when the user clicks the column header it sorts the column (if the **CanUserSortColumns** property is **True** and the **CanUserSort** property is **True** for the current column). Clicking the header of a column that is already sorted will reverse the sort direction of that column. Pressing the CTRL key while clicking multiple column headers will sort by multiple columns in the order clicked. If the property **ColumnHeaderClickAction** is set to **Select** the column will be selected if **SelectionMode** supports column selection.
CTRL+click a row	Modifies a non-contiguous multi-row selection (if **SelectionMode** support multiple rows, cells, or columns).
SHIFT+click a row	Modifies a contiguous multi-row selection (if **SelectionMode** support multiple rows, cells, or columns).

Multiple Row Selection

If the **SelectionMode** property is set to **MultiRow**, the navigation behavior does not change, but navigating with the keyboard and mouse while pressing SHIFT (including CTRL+SHIFT) will modify a multi-row selection. Before navigation starts, the control marks the current row as an anchor row. When you navigate while pressing SHIFT, the selection includes all rows between the anchor row and the current row.

Selection Keys

The following selection keys modify multi-row selection:

- SHIFT+DOWN ARROW
- SHIFT+UP ARROW
- SHIFT+PAGE DOWN
- SHIFT+PAGE UP
- CTRL+SHIFT+DOWN ARROW
- CTRL+SHIFT+UP ARROW
- CTRL+SHIFT+PAGE DOWN
- CTRL+SHIFT+PAGE UP

Mouse Selection

If the **SelectionMode** property is set to **MultiRow**, clicking a row while pressing CTRL or SHIFT will modify a multi-row selection.

When you click a row while pressing SHIFT, the selection includes all rows between the current row and an anchor row located at the position of the current row before the first click. Subsequent clicks while pressing SHIFT changes the current row, but not the anchor row.

If the CTRL key is pressed when navigating, the arrow keys will navigate to the border cells; for example, if you are in the first row and you press CTRL + DOWN you will navigate to the last row, if the SHIFT key is pressed, all the rows will be selected though.

Custom Keyboard Navigation

You can add your own custom navigation to the **C1DataGrid** control. Custom keyboard navigation enables you to control how users interact with the grid. For example, you can prevent users from navigating to read-only columns or cells with null values. In a hierarchical grid, you could set up navigation between parent and child grids. To add custom keyboard navigation you would need to handle the **KeyDown** event and then add code to override the default navigation with your customized navigation.

Adding the KeyDown Event Handler

Complete the following steps to add the **KeyDown** event handler:

1. Switch to Code view and add an event handler for the **KeyDown** event, for example:

   - Visual Basic

   ```
   Private Sub C1DataGrid1_KeyDown(ByVal sender As System.Object, ByVal e
   As System.Windows.Input.KeyEventArgs) Handles C1DataGrid1.KeyDown
       ' Add code here.
   End Sub
   ```

   - C#

   ```
   private void c1DataGrid1_KeyDown(object sender, KeyEventArgs e)
   {
       // Add code here.
   }
   ```

2. Switch to Source view and add the event handler to instances of the **C1DataGrid** control, for example:

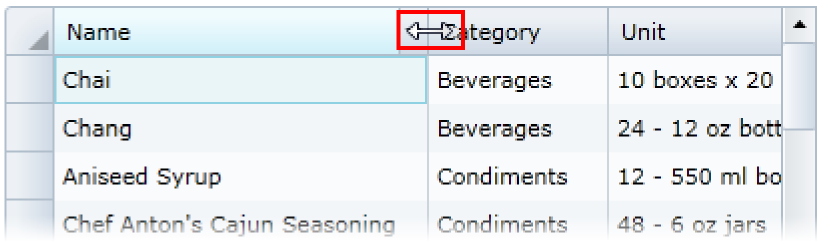
   ```
   <c1:C1DataGrid x:Name="c1DataGrid1" AutoGenerateColumns="True"
   KeyDown="c1DataGrid1_KeyDown"></c1:C1DataGrid>
   ```

You can now add code to the **KeyDown** event handler to customize the default navigation. For an example, you can take a look at the hierarchical grid example (**C1_MDSL_RowDetail**) in the **ControlExplorer** sample.

Resizing Columns and Rows

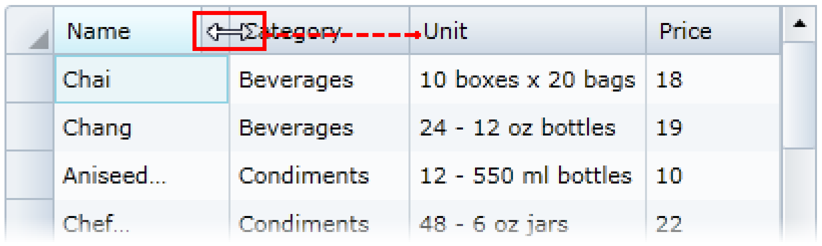
Users can easily resize columns and rows at run time through a drag-and-drop operation. To resize columns at run time, complete the following steps:

1. Navigate the mouse to the right border of a column's header. The column resizing cursor appears:



Name	ategory	Unit
Chai	Beverages	10 boxes x 20
Chang	Beverages	24 - 12 oz bott
Aniseed Syrup	Condiments	12 - 550 ml bo
Chef Anton's Cajun Seasoning	Condiments	48 - 6 oz jars

2. Click the mouse and drag the cursor to the left or the right to resize the column:



Name	ategory	Unit	Price
Chai	Beverages	10 boxes x 20 bags	18
Chang	Beverages	24 - 12 oz bottles	19
Aniseed...	Condiments	12 - 550 ml bottles	10
Chef...	Condiments	48 - 6 oz jars	22

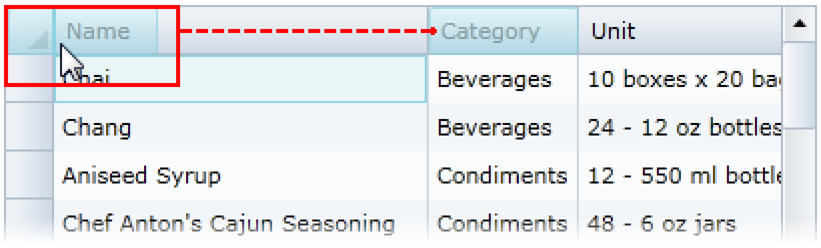
3. Release the mouse to complete the column resize operation.

Resize rows in a similar way by dragging the row indicator column. Note that the **CanUserResizeColumns** and **CanUserResizeRows** properties must be set to **True** (default) for column and row resizing to be possible. See the Disabling Column and Row Resizing (page 148) topic for more details.

Reordering Columns

End users can easily reorder columns at run time. To reorder columns at run time, complete the following steps:

1. Click the column header for the column you wish to reorder.
2. Drag the column header to where you wish the column to be ordered. Notice that a line will appear if you can place the column in that location:



Name	Category	Unit
Chai	Beverages	10 boxes x 20 ba
Chang	Beverages	24 - 12 oz bottles
Aniseed Syrup	Condiments	12 - 550 ml bottl
Chef Anton's Cajun Seasoning	Condiments	48 - 6 oz jars

3. Release the mouse to place the column in its new location and reorder the columns.

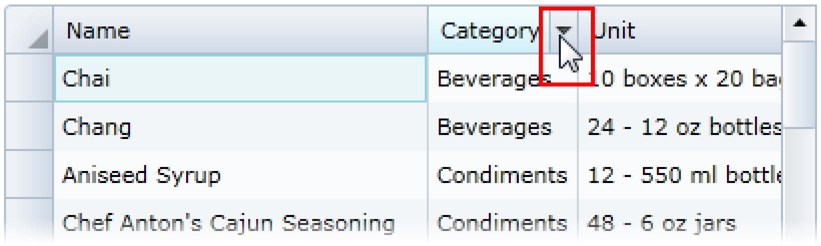
Note that the **CanUserReorderColumns** property must be set to **True** (default) for column reordering to be possible. See the Disabling Column Reordering (page 148) topic for more details.

Filtering Columns

ComponentOne DataGrid for Silverlight incorporates a filter column element in the user interface, allowing users to filter columns by specific criteria at run time.

To filter a column's text at run time, complete the following steps:

1. Click the drop-down arrow in a text column's header:



Name	Category	Unit
Chai	Beverages	10 boxes x 20 ba
Chang	Beverages	24 - 12 oz bottles
Aniseed Syrup	Condiments	12 - 550 ml bottl
Chef Anton's Cajun Seasoning	Condiments	48 - 6 oz jars

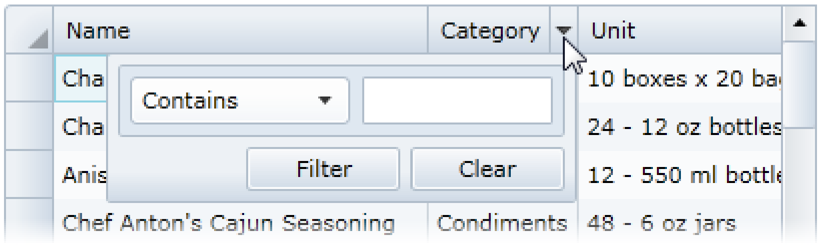
2. Enter the text in the filter text box that you want the column to be filtered by, and click the **Filter** button.

   The column will be sorted.

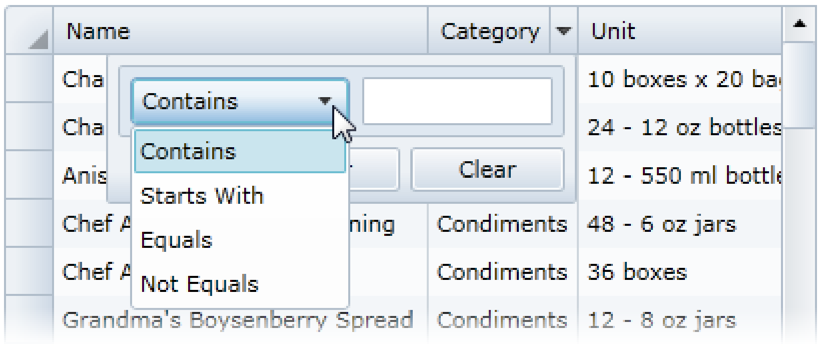
Filter options vary depending on the column type. The following filter options may be included:

- **Text Columns**

  In text columns, the filter bar appears similar to the following:

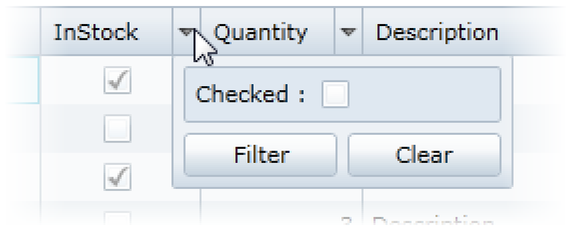


You can filter the column by whether items in the column contain, start, are equivalent to, or are not equivalent to the filter condition:



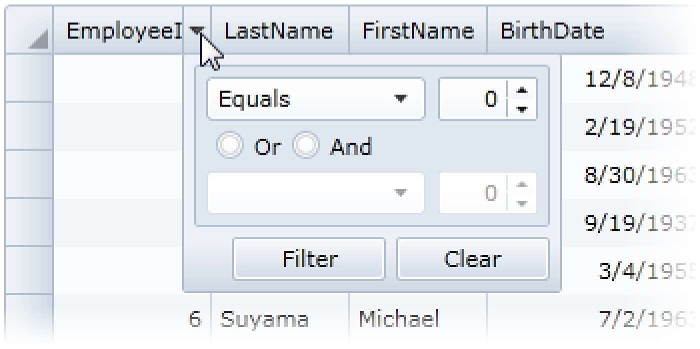
- **Boolean Columns**

  Boolean check box columns can be filtered by whether items in the column are checked or not:

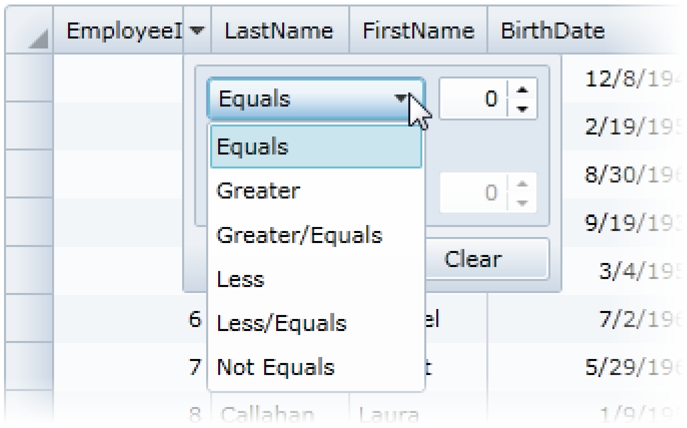


- **Numeric Columns**

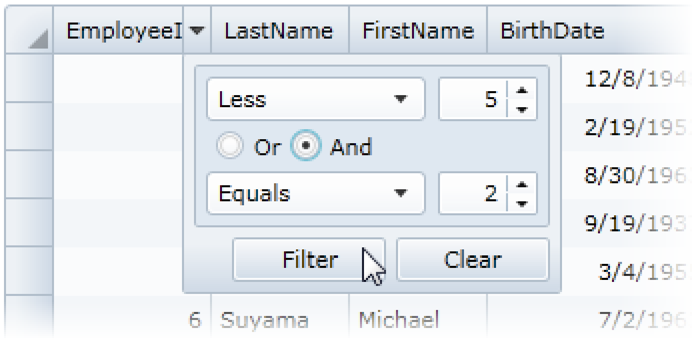
  Numeric columns offer several options for filtering:



You can filter the column by specific condition:



And you can use the **And** and **Or** radio buttons to filter by multiple conditions:

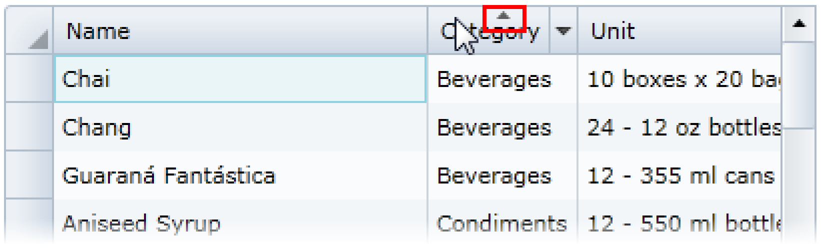


Note that the **CanUserFilter** property must be set to **True** (default) for filtering to be possible.

Sorting Columns

Sorting grid columns at run time is simple in **ComponentOne DataGrid for Silverlight**. To sort columns click once on the header of the column that you wish to sort.

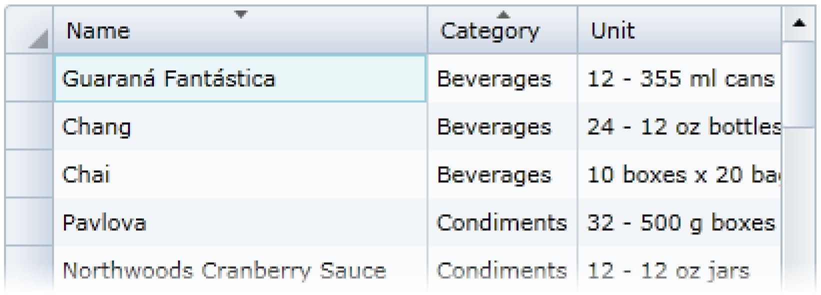
You will notice that the sort glyph, a sort direction indicator, appears when a column is sorted:



Name	Category	Unit
Chai	Beverages	10 boxes x 20 ba
Chang	Beverages	24 - 12 oz bottles
Guaraná Fantástica	Beverages	12 - 355 ml cans
Aniseed Syrup	Condiments	12 - 550 ml bottl

You can click once again on the column header to reverse the sort; notice that the sort glyph changes direction.

Sort multiple columns by sorting one column and then holding the CTRL key while clicking on a second column header to add that column to your sort condition. For example, in the following image the *Category* column was first sorted, and then the *Name* column was reverse sorted:

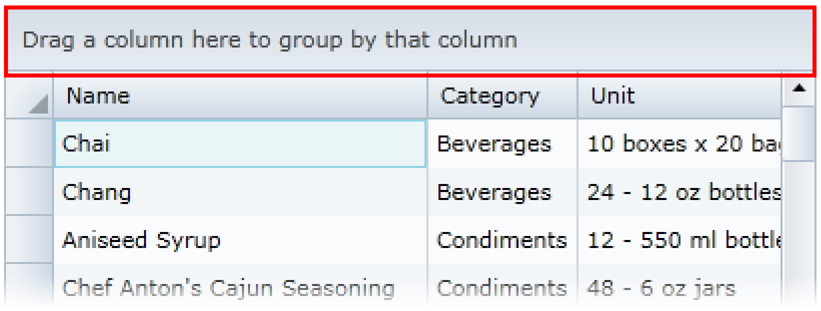


Name	Category	Unit
Guaraná Fantástica	Beverages	12 - 355 ml cans
Chang	Beverages	24 - 12 oz bottles
Chai	Beverages	10 boxes x 20 ba
Pavlova	Condiments	32 - 500 g boxes
Northwoods Cranberry Sauce	Condiments	12 - 12 oz jars

Note that the **CanUserSort** property must be set to **True** (default) for sorting to be possible.

Grouping Columns

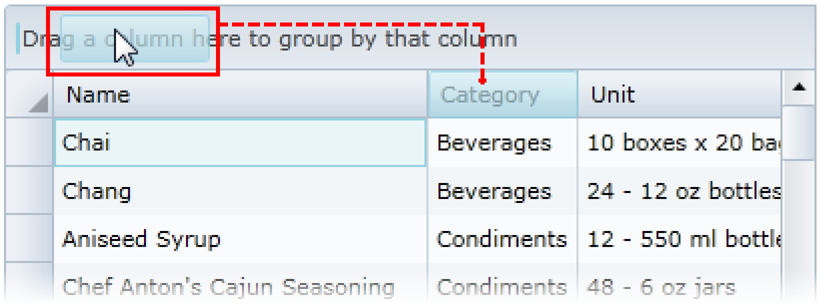
Users can group columns in your grid at run time to better organize information. The grouping area at the top of the grid allows you to easily group columns through a simple drag-and-drop operation:



Drag a column here to group by that column

Name	Category	Unit
Chai	Beverages	10 boxes x 20 ba
Chang	Beverages	24 - 12 oz bottles
Aniseed Syrup	Condiments	12 - 550 ml bottl
Chef Anton's Cajun Seasoning	Condiments	48 - 6 oz jars

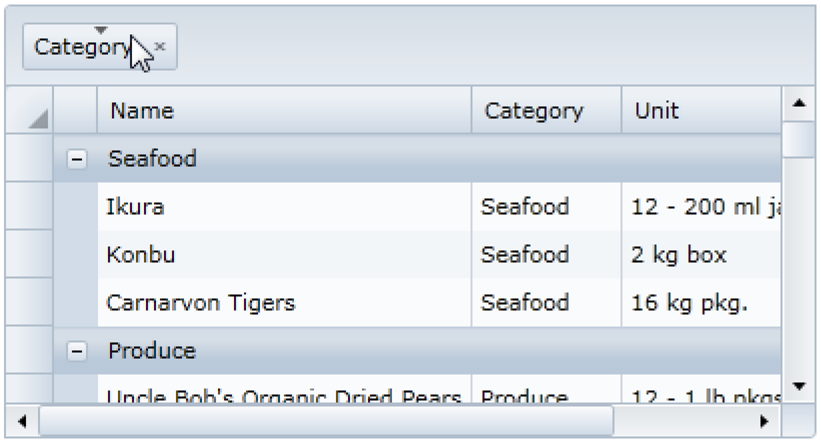
To group a column, drag a column header onto the grouping area:



Drag a column here to group by that column

Name	Category	Unit
Chai	Beverages	10 boxes x 20 ba
Chang	Beverages	24 - 12 oz bottles
Aniseed Syrup	Condiments	12 - 550 ml bottle
Chef Anton's Cajun Seasoning	Condiments	48 - 6 oz jars

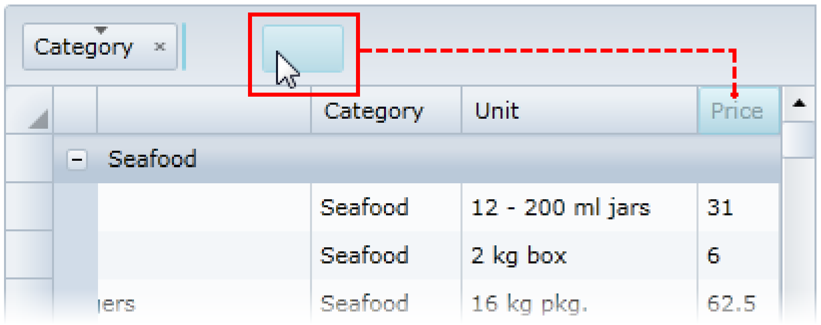
You can sort the display of grouped items, by clicking the column header in the grouping area. In the following image the grouped column has been reverse sorted:



Category ×

Name	Category	Unit
Seafood		
Ikura	Seafood	12 - 200 ml j
Konbu	Seafood	2 kg box
Carnarvon Tigers	Seafood	16 kg pkg.
Produce		
Uncle Bob's Organic Dried Pears	Produce	12 - 1 lb pkgs

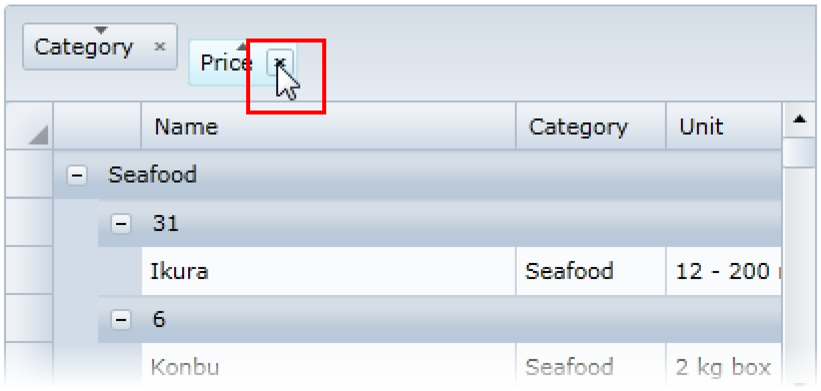
You can group multiple columns by performing a drag-and-drop operation to drag additional columns to the grouping area:



Category ×

	Category	Unit	Price
Seafood			
	Seafood	12 - 200 ml jars	31
	Seafood	2 kg box	6
ɪers	Seafood	16 kg pkg.	62.5

To remove the grouping, simply click the **X** button next to a grouped column in the grouping area of the grid:

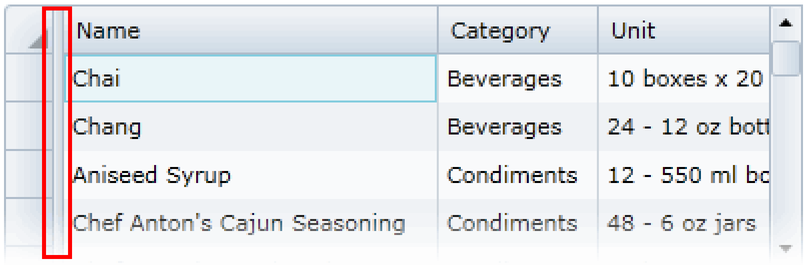


Name	Category	Unit
⊟ Seafood		
⊟ 31		
Ikura	Seafood	12 - 200
⊟ 6		
Konbu	Seafood	2 kg box

Note that the **CanUserGroup** property must be set to **True** for the grouping area to be visible and grouping to be possible (by default it is set to **False**). For more information, see Enabling Grouping in the Grid (page 146). For more information about showing the grouping area, see the Showing the Grouping Area (page 147) topic.

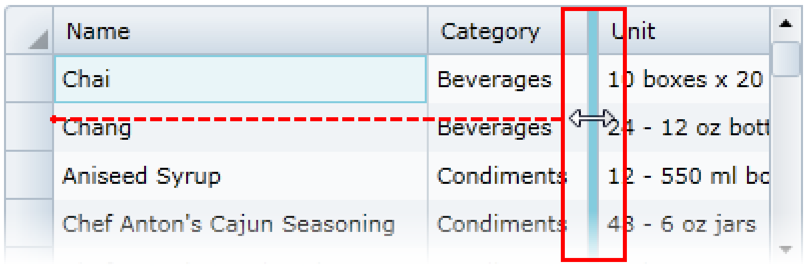
Freezing Columns

Users can freeze columns at run time to prevent them from being scrolled horizontally. This is useful as it keeps specific columns visible when the grid is resized or scrolled. The freeze bar enables users to freeze columns. When visible, the freeze bar appears to the left of the first columns by default:



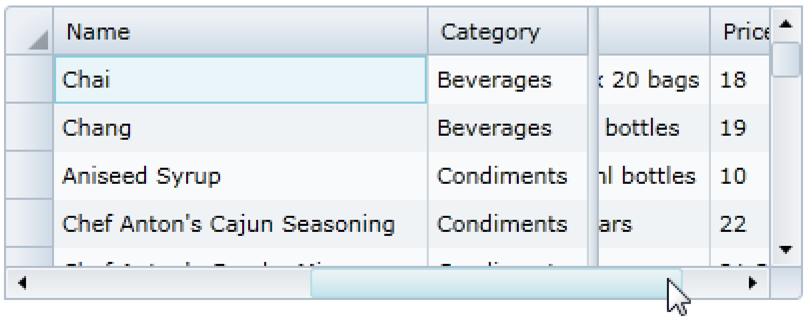
Name	Category	Unit
Chai	Beverages	10 boxes x 20
Chang	Beverages	24 - 12 oz bott
Aniseed Syrup	Condiments	12 - 550 ml bo
Chef Anton's Cajun Seasoning	Condiments	48 - 6 oz jars

To freeze specific columns, move the freeze bar to the right of the column(s) you want to freeze. For example, in the following image the freeze bar was moved to the right of the second columns:



Name	Category	Unit
Chai	Beverages	10 boxes x 20
Chang	Beverages	24 - 12 oz bott
Aniseed Syrup	Condiments	12 - 550 ml bo
Chef Anton's Cajun Seasoning	Condiments	48 - 6 oz jars

Once columns are frozen, they are not scrolled when the grid is scrolled horizontally. For example, in the following image the first two columns are frozen:



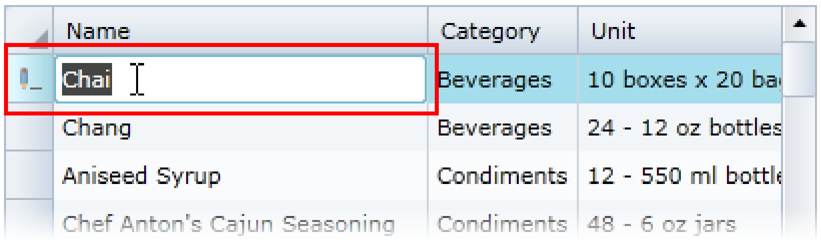
Name	Category		Price
Chai	Beverages	x 20 bags	18
Chang	Beverages	bottles	19
Aniseed Syrup	Condiments	nl bottles	10
Chef Anton's Cajun Seasoning	Condiments	ars	22

Note that the **ShowVerticalFreezingSeparator** property must be set to **Left** (by default **None**) for the freeze bar to be visible and the **CanUserFreezeColumns** property must be set to **Left** (by default **None**) to allow users to freeze columns are run time. See Enabling Column Freezing (page 150) for an example.

Editing Cells

Users can easily edit cell content at run time. Editing content is as simple as selecting a cell and deleting or changing the content in that cell. Complete the following steps to edit cell content:

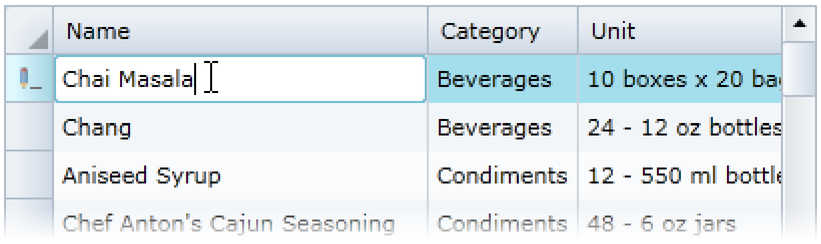
1. Double-click the cell you would like to edit.



Name	Category	Unit
Chai	Beverages	10 boxes x 20 ba
Chang	Beverages	24 - 12 oz bottles
Aniseed Syrup	Condiments	12 - 550 ml bottl
Chef Anton's Cajun Seasoning	Condiments	48 - 6 oz jars

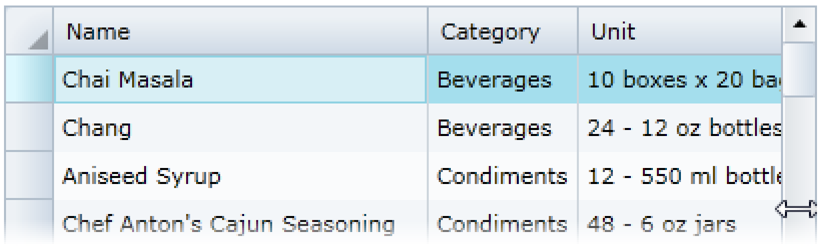
A cursor will appear in that cell indicating that it can be edited and a pencil icon will appear in the row indicator column, indicating that a cell in that row is in edit mode.

2. Delete text or type in new or additional text to edit the content of the cell:



Name	Category	Unit
Chai Masala	Beverages	10 boxes x 20 ba
Chang	Beverages	24 - 12 oz bottles
Aniseed Syrup	Condiments	12 - 550 ml bottl
Chef Anton's Cajun Seasoning	Condiments	48 - 6 oz jars

3. Press ENTER or click away from the cell you are editing for the changes you made to take effect:



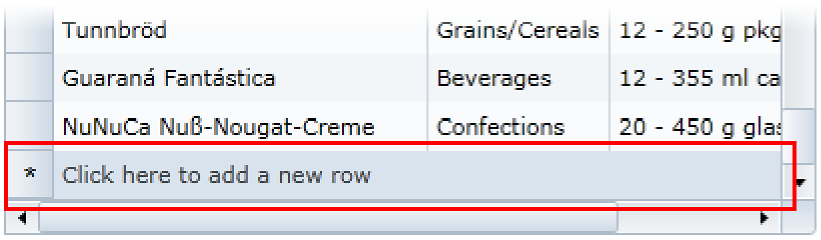
Name	Category	Unit
Chai Masala	Beverages	10 boxes x 20 ba
Chang	Beverages	24 - 12 oz bottles
Aniseed Syrup	Condiments	12 - 550 ml bottl
Chef Anton's Cajun Seasoning	Condiments	48 - 6 oz jars

The pencil icon indicating editing will no longer be visible.

Note that the **CanUserEditRows** property must be set to **True** (default) for editing to be possible. See Disabling Cell Editing (page 151) for an example.

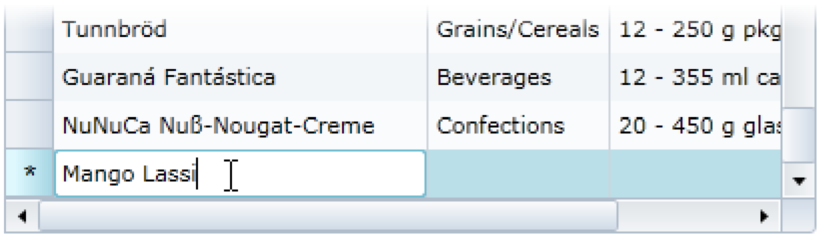
Adding Rows to the Grid

You can add rows to the grid at run time using the new row bar. The new row bar, located at the bottom of the grid by default and indicated by an asterisk symbol (*), allows you to type in new information to add to the grid at run time:



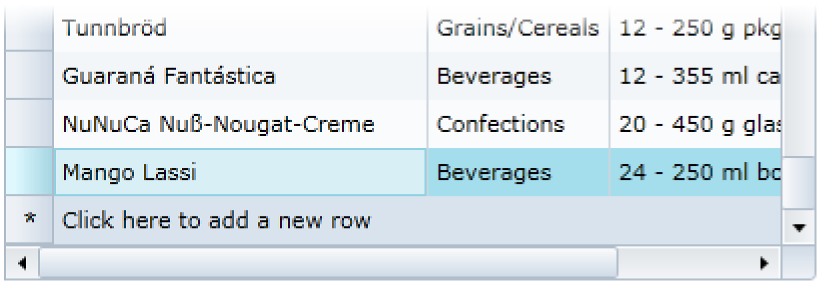
	Tunnbröd	Grains/Cereals	12 - 250 g pkg
	Guaraná Fantástica	Beverages	12 - 355 ml ca
	NuNuCa Nuß-Nougat-Creme	Confections	20 - 450 g glas
*	Click here to add a new row		

To add a new row, simply type text into the new row bar:



	Tunnbröd	Grains/Cereals	12 - 250 g pkg
	Guaraná Fantástica	Beverages	12 - 355 ml ca
	NuNuCa Nuß-Nougat-Creme	Confections	20 - 450 g glas
*	Mango Lassi		

Press ENTER for text to be added to the grid in a new row:



	Tunnbröd	Grains/Cereals	12 - 250 g pkg
	Guaraná Fantástica	Beverages	12 - 355 ml ca
	NuNuCa Nuß-Nougat-Creme	Confections	20 - 450 g glas
	Mango Lassi	Beverages	24 - 250 ml bo
*	Click here to add a new row		

Note that the **CanUserAddRows** property must be set to **True** (default) for row adding to be possible. See Disabling Adding Rows (page 152) for an example.

DataGrid for Silverlight Tutorials

The following tutorials provide additional information about **ComponentOne DataGrid for Silverlight** and walk through steps to further customize your grid. Tutorials include additional data binding, styling, and behavior customization steps and highlight advanced features.

Binding the Grid to a Web Service

The following tutorial will walk you through the process of binding the **C1DataGrid** control to the standard Northwind database and creating a Web Service.

Step 1 of 3: Creating the User Interface

In this step you'll begin in Visual Studio to create a Silverlight grid application. You'll then continue by creating and customizing the application's user interface (UI) and adding the **C1DataGrid** control to your project.

To set up your project, complete the following steps:

1. In Visual Studio, select **File | New | Project**.
2. In the **New Project** dialog box, select a language in the left pane and in the templates list select **Silverlight Application**. Enter a **Name** for your project, for example "ComponentOneDataGrid", and click **OK**. The **New Silverlight Application** dialog box will appear.
3. Click **OK** to accept the default settings, close the **New Silverlight Application** dialog box, and create your project.
4. If the **MainPage.xaml** file is not currently open, navigate to the Solution Explorer and double-click on the **MainPage.xaml** item.
5. In the XAML view, locate the `<UserControl>` tag.
6. In the `<UserControl>` tag, replace `Width="400" Height="300"` (or `d:DesignWidth="400" d:DesignHeight="300"`) with `Width="600" Height="400"`.

   This will increase the size of your Silverlight application.
7. Place the cursor just after the `<Grid x:Name="LayoutRoot" Background="White">` tag and add the following markup:
   ```
   <!-- Grid Layout-->
   <Grid.RowDefinitions>
   <RowDefinition Height="Auto" />
   <RowDefinition Height="*" />
   <RowDefinition Height="Auto" />
   </Grid.RowDefinitions>
   ```
   This row definition will define the layout of your grid.
8. Add a title to your application by adding the following **TextBlock** just under the `</Grid.RowDefinitions>` tag:
   ```
   <!-- Title -->
   <TextBlock Text="ComponentOne DataGrid for Silverlight" Margin="5"
   FontSize="16"/>
   ```
9. In the XAML window of the project, place the cursor just above the `</Grid>` tag and click once.
10. Navigate to the Toolbox and double-click the **C1DataGrid** icon to add the grid control to **MainPage.xaml**. The XAML markup will now look similar to the following:

```
<UserControl xmlns:c1="http://schemas.componentone.com/winfx/2006/xaml"
x:Class="C1DataGrid.MainPage"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml" Width="600"
Height="400">
    <Grid x:Name="LayoutRoot" Background="White">
       <!-- Grid Layout-->
       <Grid.RowDefinitions>
           <RowDefinition Height="Auto" />
           <RowDefinition Height="*" />
           <RowDefinition Height="Auto" />
       </Grid.RowDefinitions>
       <!-- Title -->
       <TextBlock Text="ComponentOne DataGrid for Silverlight"
Margin="5" FontSize="16"/>
        <c1:C1DataGrid></c1:C1DataGrid>
    </Grid>
</UserControl>
```

Note that the **C1.Silverlight.DataGrid** namespace and `<c1:C1DataGrid></c1:C1DataGrid>` tags have been added to the project.

11. If the `<c1:C1DataGrid>` tag includes existing content, delete it so it appears similar to the following:

    ```
    <c1:C1DataGrid>
    ```

12. Give your grid a name by adding `x:Name="_c1DataGrid"` to the `<c1:C1DataGrid>` tag so that it appears similar to the following:

    ```
    <c1:C1DataGrid x:Name="_c1DataGrid">
    ```

    By giving the control a unique identifier, you'll be able to access the **C1DataGrid** control in code.

13. Define the location of your grid by adding `Grid.Row="1"` to the `<c1:C1DataGrid>` tag so that it appears similar to the following:

    ```
    <c1:C1DataGrid x:Name="_c1DataGrid" Grid.Row="1">
    ```

14. Add the following markup just after the `</c1:C1DataGrid>` tag:

    ```
    <TextBlock x:Name="_tbStatus" Text="Ready"
       VerticalAlignment="Center" FontSize="12" Foreground="Gray" Margin="5"
    Grid.Row="2" />
    ```

    This **TextBlock** will be used to display status information text.

What You've Accomplished

Run your application, and observe that your page includes a title, a grid, and text below the grid. You've successfully created a basic grid application, but the grid is blank and contains no data. In the next steps you'll add a database to your project and bind the grid to a data source.

Step 2 of 3: Adding a Database and Web Service

In this step you'll add a database to your project, and begin the process of binding the grid. Note that in this step you'll be using the standard Northwind database and an example code file, both of which should be installed with the **Studio for Silverlight** samples.

To set up your project, complete the following steps:

1. In the Solution Explorer, expand the .Web project (for example ComponentOneDataGrid.Web) and if the **App_Data** folder is not visible, right click the .Web project, and select **Add | New Folder**. Name the folder "App_Data".
2. In the Solution Explorer, right click the **App_Data** node, and select **Add | Existing Item**.

3. In the **Add Existing Item** dialog box, navigate to the **ComponentOne Samples\Studio for Silverlight\C1.Silverlight.DataGrid\C1DataGrid_MDSL\C1_MDSLWeb\App_Data** directory within the **Documents** or **My Documents** folder, select the **NWIND.mdb** file, and click **Add** to add it to your project.
4. In the Solution Explorer, select the **NWIND.MDB** file you just added, and in the Properties window set its **Build Action** property to **None**
5. In the Solution Explorer, right-click the .Web project (for example ComponentOneDataGrid.Web) and select **Add | Existing Item**.
6. In the **Add Existing Item** dialog box, navigate to the **ComponentOne Samples\Studio for Silverlight\C1.Silverlight.DataGrid\C1DataGrid_MDSL\C1_MDSLWeb** directory within the **Documents** or **My Documents** folder, select the **SmartDataSet.cs** file, and click **Add** to add it to your project.

   Note that for Visual Basic projects, you can find the **SmartDataSet.vb** file posted online in the forums. This file contains code allowing data transfer to and from the database.
7. In the Solution Explorer, right-click the .Web project and select **Add | New Item**.
8. In the left pane of the **Add New Item** dialog box, select the **Web** item.
9. In the templates list, select **Web Service**, name the Web Service "DataService.asmx", and click the **Add** button. Note that the Web Service file will be added to your project and automatically opened.
10. In the **DataService.asmx** file, add the following using statements at the top of the file:
    - Visual Basic
    ```
    Imports System.IO
    Imports System.Data
    Imports C1_MDSLWeb ' SmartDataSet namespace
    ```
    - C#
    ```
    using System.IO;
    using System.Data;
    using C1_MDSLWeb; // SmartDataSet namespace
    ```
11. Next, uncomment the `[System.Web.Script.Services.ScriptService]` or `<System.Web.Script.Services.ScriptService()>` line.

    This will allow the Web Service to be called from script.
12. Delete the existing **HelloWorld** method and replace it with the following code:
    - Visual Basic
    ```
    <WebMethod> _
    Public Function GetData(tables As String) As Byte()
       ' Create DataSet with connection string
       Dim ds = GetDataSet()
       ' Load data into DataSet
       ds.Fill(tables.Split(","C))
       ' Persist to stream
       Dim ms = New System.IO.MemoryStream()
       ds.WriteXml(ms, XmlWriteMode.WriteSchema)
       ' Return stream data
       Return ms.ToArray()
    End Function
    Private Function GetDataSet() As SmartDataSet
       ' Get physical location of the mdb file
    ```

```
   Dim mdb As String =
Path.Combine(Context.Request.PhysicalApplicationPath,
"App_Data\nwind.mdb")
   ' Check that the file exists
   If Not File.Exists(mdb) Then
         Dim msg As String = String.Format("Cannot find database file
{0}.", mdb)
         Throw New FileNotFoundException(msg)
   End If
   ' Make sure file is not read-only (source control often does
this...)
   Dim att As FileAttributes = File.GetAttributes(mdb)
   If (att And FileAttributes.[ReadOnly]) <> 0 Then
         att = att And Not FileAttributes.[ReadOnly]
         File.SetAttributes(mdb, att)
   End If
   ' Create and initialize the SmartDataSet
   Dim dataSet = New SmartDataSet()
   dataSet.ConnectionString = "provider=microsoft.jet.oledb.4.0;data
source=" & mdb
   Return dataSet
End Function
```

- C#

```
[WebMethod]
public byte[] GetData(string tables)
{
    // Create DataSet with connection string
    var ds = GetDataSet();
    // Load data into DataSet
    ds.Fill(tables.Split(','));
    // Persist to stream
    var ms = new System.IO.MemoryStream();
    ds.WriteXml(ms, XmlWriteMode.WriteSchema);
    // Return stream data
    return ms.ToArray();
}
SmartDataSet GetDataSet()
{
    // Get physical location of the mdb file
    string mdb = Path.Combine(
    Context.Request.PhysicalApplicationPath, @"App_Data\nwind.mdb");
    // Check that the file exists
    if (!File.Exists(mdb))
    {
        string msg = string.Format("Cannot find database file {0}.",
mdb);
        throw new FileNotFoundException(msg);
    }
    // Make sure file is not read-only (source control often does
this...)
    FileAttributes att = File.GetAttributes(mdb);
    if ((att & FileAttributes.ReadOnly) != 0)
    {
        att &= ~FileAttributes.ReadOnly;
        File.SetAttributes(mdb, att);
    }
    // Create and initialize the SmartDataSet
    var dataSet = new SmartDataSet();
    dataSet.ConnectionString = "provider=microsoft.jet.oledb.4.0;data
source=" + mdb;
    return dataSet;
}
```

This code will create a dataset and take data from the database.

13. Right-click the .Web project (for example ComponentOneDataGrid.Web) and select **Build** from the context menu. Note that you'll now be done with the ComponentOneDataGrid.Web project and will return to working with the ComponentOneDataGrid project.

✓ **What You've Accomplished**

In this step you've added a database to your project and created a Web Service. In the next step you'll finish connecting the Web Service to your project and you'll run your application.

Step 3 of 3: Connecting the Web Service

In the previous step you created a Web Service and added a database to your project. In this step you'll continue by linking the Web Service to your application. Note that this step requires **ComponentOne Data for Silverlight**.

To set up your project, complete the following steps:

1. In the Solution Explorer, expand the project's node, right-click the project name (for example ComponentOneDataGrid) and select **Add Reference** from the context menu.
2. In the **Add Reference** dialog box, add a reference to the **C1.Silverlight.Data** assembly and click **OK**.
3. In the Solution Explorer, right-click the project name and select **Add Service Reference** from the context menu.
4. In the **Add Service Reference** dialog box click the **Discover** button. The DataService.asmx file will appear in the list of Services.
5. In the **Namespace** text box, change the default value to "DataService" and click the **OK** button to save your settings and close the dialog box.
6. In the Solution Explorer, expand the **MainPage.xaml** node and double-click the **MainPage.xaml.cs** or **MainPage.xaml.vb** file to open it in the Code Editor.
7. Add the following import statements at the top of the file:

   - Visual Basic

```
Imports System.IO
Imports C1.Silverlight.Data
Imports ComponentOneDataGrid.DataService ' ComponentOneDataGrid is the
project's namespace, change this if the name of your project is
different.
```

   - C#

```
using System.IO;
using C1.Silverlight.Data;
using ComponentOneDataGrid.DataService; // ComponentOneDataGrid is the
project's namespace, change this if the name of your project is
different.
```

8. Add `LoadData();` to the **MainPage** constructor so it appears like the following:

   - Visual Basic

```
Public Sub New()
    InitializeComponent()
    LoadData()
```

```
End Sub
```

- C#

```
public MainPage()
{
    InitializeComponent();
    LoadData();
}
```

9. Add the **LoadData** and **svc_GetDataCompleted** methods to retrieve data from the Web Service:

- Visual Basic

```
Private _ds As DataSet = Nothing
Private Sub LoadData()
   ' Invoke Web Service
   Dim svc = GetDataService()
   AddHandler svc.GetDataCompleted, AddressOf svc_GetDataCompleted
   'svc.GetDataAsync("Categories,Products,Employees");
   svc.GetDataAsync("Employees")
End Sub
Private Sub svc_GetDataCompleted(sender As Object, e As
GetDataCompletedEventArgs)
   ' Handle errors
   If e.[Error] IsNot Nothing Then
         _tbStatus.Text = "Error downloading data..."
         Return
   End If
   ' Parse data stream from server (DataSet as XML)
   _tbStatus.Text = String.Format("Got data, {0:n0} kBytes",
e.Result.Length / 1024)
   Dim ms = New MemoryStream(e.Result)
   _ds = New DataSet()
   _ds.ReadXml(ms)
   ' Bind control to the data
   BindData()
End Sub
```

- C#

```
DataSet _ds = null;
void LoadData()
{
    // Invoke Web Service
    var svc = GetDataService();
    svc.GetDataCompleted += svc_GetDataCompleted;
    //svc.GetDataAsync("Categories,Products,Employees");
    svc.GetDataAsync("Employees");
}
void svc_GetDataCompleted(object sender, GetDataCompletedEventArgs e)
{
    // Handle errors
    if (e.Error != null)
    {
        _tbStatus.Text = "Error downloading data...";
        return;
    }
    // Parse data stream from server (DataSet as XML)
    _tbStatus.Text = string.Format("Got data, {0:n0} kBytes",
e.Result.Length / 1024);
```

```
    var ms = new MemoryStream(e.Result);
    _ds = new DataSet();
    _ds.ReadXml(ms);
    // Bind control to the data
    BindData();
}
```

10. Implement the **GetDataService()** method by adding the following code:

- Visual Basic

```
' Get data service relative to current host/domain
Private Function GetDataService() As DataServiceSoapClient
   ' Increase buffer size
   Dim binding = New System.ServiceModel.BasicHttpBinding()
   binding.MaxReceivedMessageSize = 2147483647
   ' int.MaxValue
   binding.MaxBufferSize = 2147483647
   ' int.MaxValue
   ' Get absolute service address
   Dim uri As Uri =
C1.Silverlight.Extensions.GetAbsoluteUri("DataService.asmx")
   Dim address = New System.ServiceModel.EndpointAddress(uri)
   ' Return new service client
   Return New DataServiceSoapClient(binding, address)
End Function
```

- C#

```
// Get data service relative to current host/domain
DataServiceSoapClient GetDataService()
{
    // Increase buffer size
    var binding = new System.ServiceModel.BasicHttpBinding();
    binding.MaxReceivedMessageSize = 2147483647;
    // int.MaxValue
    binding.MaxBufferSize = 2147483647;
    // int.MaxValue
    // Get absolute service address
    Uri uri =
C1.Silverlight.Extensions.GetAbsoluteUri("DataService.asmx");
    var address = new System.ServiceModel.EndpointAddress(uri);
    // Return new service client
    return new DataServiceSoapClient(binding, address);
}
```

11. Implement the **BindData()** method by adding the following code:

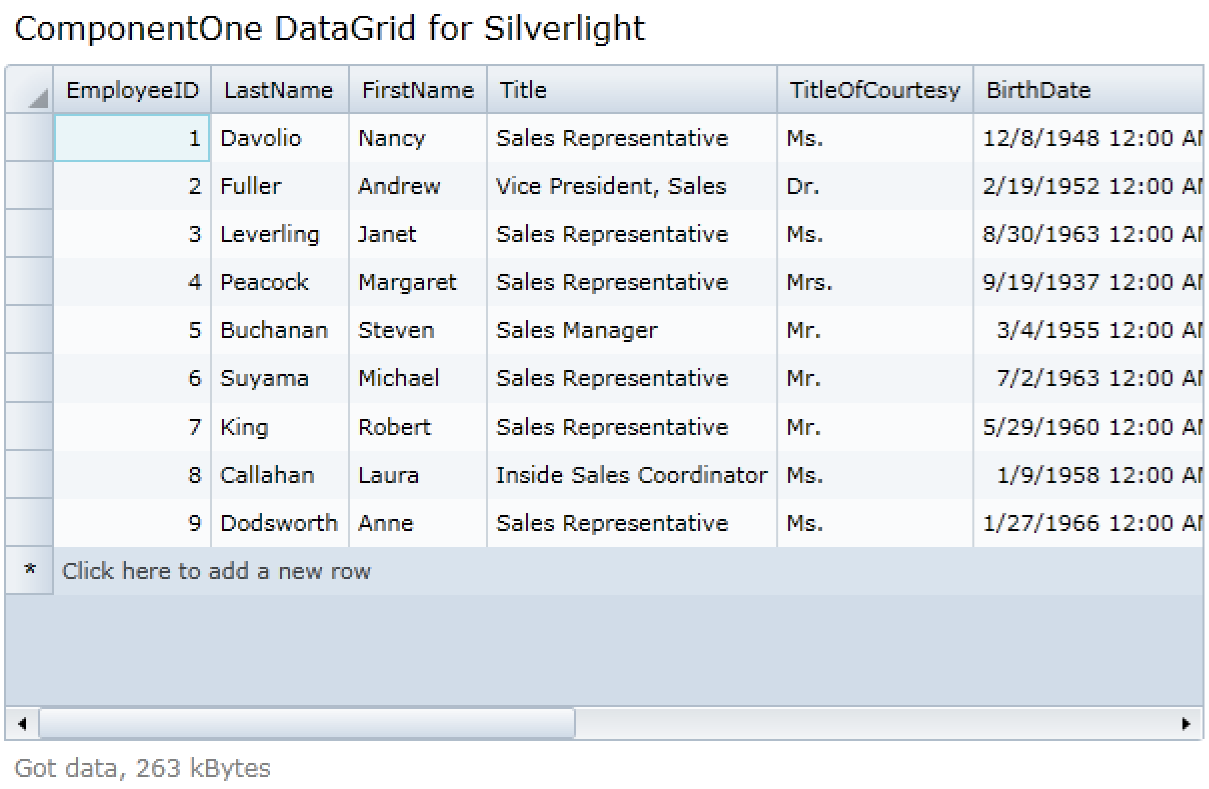
- Visual Basic

```
Private Sub BindData()
   ' Get the tables
   Dim dtEmployees As DataTable = _ds.Tables("Employees")
   ' Populate categories grid
   _c1DataGrid.ItemsSource = dtEmployees.DefaultView
End Sub
```

- C#

```
void BindData()
{
    // Get the tables
    DataTable dtEmployees = _ds.Tables["Employees"];
    // Populate categories grid
    _c1DataGrid.ItemsSource = dtEmployees.DefaultView;
}
```

12. Run your application and observe that the grid appears bound to the *Employees* table of the Northwind database:



ComponentOne DataGrid for Silverlight

EmployeeID	LastName	FirstName	Title	TitleOfCourtesy	BirthDate
1	Davolio	Nancy	Sales Representative	Ms.	12/8/1948 12:00 AI
2	Fuller	Andrew	Vice President, Sales	Dr.	2/19/1952 12:00 AI
3	Leverling	Janet	Sales Representative	Ms.	8/30/1963 12:00 AI
4	Peacock	Margaret	Sales Representative	Mrs.	9/19/1937 12:00 AI
5	Buchanan	Steven	Sales Manager	Mr.	3/4/1955 12:00 AI
6	Suyama	Michael	Sales Representative	Mr.	7/2/1963 12:00 AI
7	King	Robert	Sales Representative	Mr.	5/29/1960 12:00 AI
8	Callahan	Laura	Inside Sales Coordinator	Ms.	1/9/1958 12:00 AI
9	Dodsworth	Anne	Sales Representative	Ms.	1/27/1966 12:00 AI

* Click here to add a new row

Got data, 263 kBytes

What You've Accomplished

Congratulations, you've completed this tutorial! In this tutorial you created a new Silverlight project, added an Access database, and created a Web Service to bind the **C1DataGrid** control.

Binding the Grid to an RSS Feed

The following tutorial will walk you through the process of binding the **C1DataGrid** control to an RSS feed. Note that in this example, you will be binding the grid to the ComponentOne Buzz RSS news feed at http://helpcentral.componentone.com/CS/blogs/c1buzz/rss.aspx . Note that you can substitute another RSS feed in the steps below, if you choose.

Complete the following steps:

1. In Visual Studio, select **File | New | Project**.
2. In the **New Project** dialog box, select a language in the left pane and in the templates list select **Silverlight Application**. Enter a **Name** for your project, for example "C1DataGridRSS", and click **OK**. The **New Silverlight Application** dialog box will appear.
3. Click **OK** to accept the default settings, close the **New Silverlight Application** dialog box, and create your project.
4. In the Solution Explorer window, right-click the project name (for example, C1DataGridRSS) and select **Add Reference**.

5. In the **Add Reference** dialog box, locate the **System.Xml.Linq** library and click **OK** to add a reference to your project.

6. In the XAML window of the project, place the cursor between the `<Grid>` and `</Grid>` tags and click once.

7. Navigate to the Toolbox and double-click the **C1DataGrid** icon to add the grid control to **MainPage.xaml**. The XAML markup will now look similar to the following:

   ```
   <UserControl xmlns:c1="http://schemas.componentone.com/winfx/2006/xaml"
   x:Class="C1DataGrid.MainPage"
   xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
   xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml" Width="400"
   Height="300">
       <Grid x:Name="LayoutRoot" Background="White">
           <c1:C1DataGrid></c1:C1DataGrid>
       </Grid>
   </UserControl>
   ```

   Note that the C1.Silverlight.DataGrid namespace and `<c1:C1DataGrid></c1:C1DataGrid>` tags have been added to the project.

8. If the `<c1:C1DataGrid>` tag includes existing content, delete it so it appears similar to the following:

   ```
   <c1:C1DataGrid></c1:C1DataGrid>
   ```

9. Give your grid a name by adding `x:Name="c1grid"` to the `<c1:C1DataGrid>` tag so that it appears similar to the following:

   ```
   <c1:C1DataGrid x:Name="c1grid">
   ```

   By giving the control a unique identifier, you'll be able to access the **C1DataGrid** control in code.

10. Add `AutoGenerateColumns="True"` to the `<c1:C1DataGrid>` tag so that it appears similar to the following:

    ```
    <c1:C1DataGrid x:Name="c1grid" AutoGenerateColumns="True">
    ```

    This way the grid will automatically generate and display data from the data source.

11. In the Solution Explorer, expand the **MainPage.xaml** node and double-click the **MainPage.xaml.cs** (or **MainPage.xaml.vb**) file to open it in the Code Editor.

12. Add the following import statement at the top of the file:

    - Visual Basic

      ```
      Imports System.Xml.Linq
      ```

    - C#

      ```
      using System.Xml.Linq;
      ```

13. In the **MainPage** constructor, add an event handler and set up a **WebClient** object to read from the RSS feed with the following code:

    - Visual Basic

      ```
      Public Sub New()
          InitializeComponent()
          Dim client As New WebClient()
          Dim uri As New
      Uri("http://helpcentral.componentone.com/CS/blogs/c1buzz/rss.aspx")
          AddHandler client.DownloadStringCompleted, AddressOf
      client_DownloadStringCompleted
          client.DownloadStringAsync(uri)
      End Sub
      ```

    - C#

      ```
      public MainPage()
      ```

```
{
    InitializeComponent();
    WebClient client = new WebClient();
    Uri uri = new
Uri("http://helpcentral.componentone.com/CS/blogs/c1buzz/rss.aspx");
    client.DownloadStringCompleted += new
DownloadStringCompletedEventHandler(client_DownloadStringCompleted);
    client.DownloadStringAsync(uri);
}
```

Note that you can substitute another RSS feed for the ComponentOne Buzz feed, if you choose.

14. Add the **News** class:

    - Visual Basic

```
Public Class News
    Public Property Title() As String
        Get
            Return m_Title
        End Get
        Set(ByVal value As String)
            m_Title = Value
        End Set
    End Property
    Private m_Title As String
    Public Property Link() As String
        Get
            Return m_Link
        End Get
        Set(ByVal value As String)
            m_Link = Value
        End Set
    End Property
    Private m_Link As String
End Class
```

    - C#

```
public class News
{
    public string Title { get; set; }
    public string Link { get; set; }
}
```

15. Add the **client_DownloadStringCompleted** event handler:

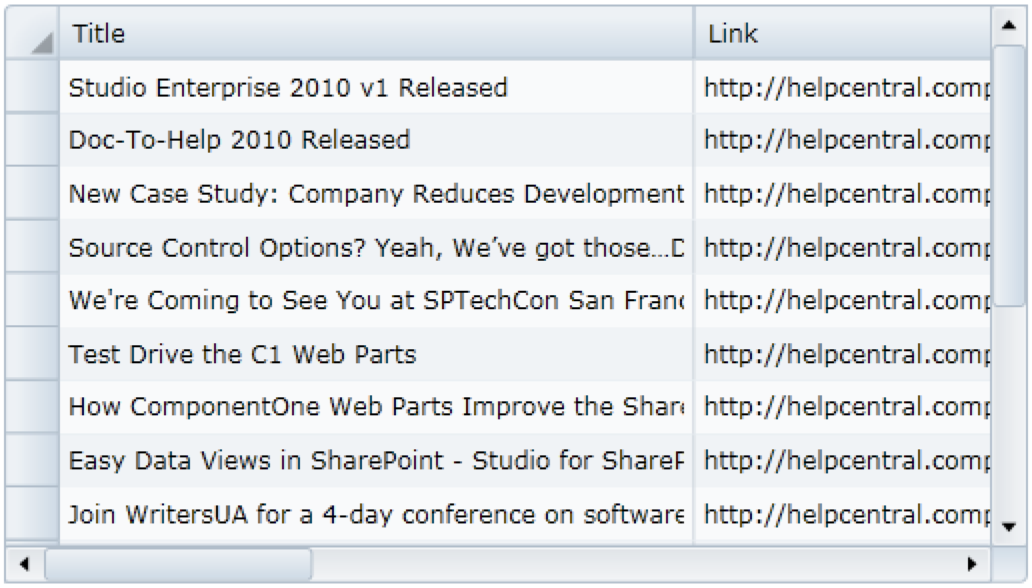
    - Visual Basic

```
Private Sub client_DownloadStringCompleted(ByVal sender As Object,
ByVal e As DownloadStringCompletedEventArgs)
    Dim xmlNews As XDocument = XDocument.Parse(e.Result)
    Dim news = From story In xmlNews.Descendants("item") _
    Select New News With {.Title = story.Element("title").Value, .Link
= story.Element("link").Value}
    c1grid.ItemsSource = news
End Sub
```

    - C#

```
void client_DownloadStringCompleted(object sender,
DownloadStringCompletedEventArgs e)
{
    XDocument xmlNews = XDocument.Parse(e.Result);
    var news = from story in xmlNews.Descendants("item")
        select new News
        {
           Title = (string)story.Element("title"),
           Link = (string)story.Element("link")
        };
    c1grid.ItemsSource = news;
}
```

16. Run your application and observe that the grid appears bound to the ComponentOne Buzz RSS news feed:



Title	Link
Studio Enterprise 2010 v1 Released	http://helpcentral.comp
Doc-To-Help 2010 Released	http://helpcentral.comp
New Case Study: Company Reduces Development	http://helpcentral.comp
Source Control Options? Yeah, We’ve got those...D	http://helpcentral.comp
We're Coming to See You at SPTechCon San Franc	http://helpcentral.comp
Test Drive the C1 Web Parts	http://helpcentral.comp
How ComponentOne Web Parts Improve the Shar	http://helpcentral.comp
Easy Data Views in SharePoint - Studio for ShareP	http://helpcentral.comp
Join WritersUA for a 4-day conference on software	http://helpcentral.comp

What You've Accomplished

Congratulations, you've completed this tutorial! In this topic you created a new Silverlight project, added a **C1DataGrid** control, and learned how to bind the grid to an RSS feed.

Creating a Master/Detail View

The following tutorial will walk you through using the **C1DataGrid** control to present data in a master/detail view using the row details feature.

The following example shows a set of product categories loaded from a XML file using LINQ to XML. For each row in the main grid (categories), a list of products is loaded and shown in the detail view using a second **C1DataGrid** control. The detail data is loaded when the detail view of a category row changes.

For more information, see the **C1DataGrid_Demo** sample installed with **ComponentOne Studio for Silverlight**.

Step 1 of 3: Setting up the Master/Detail Grid

In this step you'll begin in Visual Studio to create a Silverlight grid application using **ComponentOne DataGrid for Silverlight**. You'll create a new Silverlight project and add the **C1DataGrid** control to your application.

To set up your project and add a **C1DataGrid** control to your application, complete the following steps:

1. In Visual Studio, select **File | New | Project**.

2. In the **New Project** dialog box, select a language in the left pane (in this example, **C#** is used), and in the templates list select **Silverlight Application**. Enter "MasterDetail" in the **Name** text box, and click **OK**. The **New Silverlight Application** dialog box will appear.

3. Click **OK** to accept the default settings, close the **New Silverlight Application** dialog box, and create your project.

4. In the XAML window of the project, place the cursor between the `<Grid>` and `</Grid>` tags and click once.

5. Navigate to the Toolbox and double-click the **C1DataGrid** icon to add the grid control to **MainPage.xaml**. The XAML markup will now look similar to the following:
```
<UserControl xmlns:c1="http://schemas.componentone.com/winfx/2006/xaml"
x:Class="MasterDetail.MainPage"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
mc:Ignorable="d" d:DesignWidth="640" d:DesignHeight="480">
  <Grid x:Name="LayoutRoot">
      <c1:C1DataGrid></c1:C1DataGrid>
  </Grid>
</UserControl>
```

6. If the `<c1:C1DataGrid>` tag includes existing content, delete it so it appears similar to the following:
```
<c1:C1DataGrid></c1:C1DataGrid>
```

7. Give your grid a name by adding `x:Name="c1dg"` to the `<c1:C1DataGrid>` tag so that it appears similar to the following:
```
<c1:C1DataGrid x:Name="c1dg">
```
   By giving the control a unique identifier, you'll be able to access the **C1DataGrid** control in code.

8. Add `CanUserAddRows="False"` to the `<c1:C1DataGrid>` tag so that it appears similar to the following:
```
<c1:C1DataGrid x:Name="c1dg" CanUserAddRows="False">
```
   Users will not be able to add new rows to the grid.

9. Add `Margin="5"` to the `<c1:C1DataGrid>` tag so that it appears similar to the following:
```
<c1:C1DataGrid x:Name="c1dg" CanUserAddRows="False" Margin="5">
```
   This will add a margin around the grid.

What You've Accomplished

You've successfully created a basic grid application. In the next step you'll add a XML data source to your project.

Step 2 of 3: Adding a Data Source to the Project

In this step you'll add a data source to your application and add external files to set up the data source. Note that to simplify the tutorial, this step uses files included with the **C1DataGrid_Demo** sample included with the **Studio for Silverlight** installation; by default, **products.xml** and **Data.cs** will be installed in the **Documents** or **My Documents** folder in the **ComponentOne Samples\Studio for Silverlight\C1.Silverlight.DataGrid\C1DataGrid_Demo\C1DataGrid_Demo** directory.

To add a data source, complete the following steps:

1. In the Solution Explorer window, right-click the **MasterDetail** project and select **Add | New Folder**. Rename the folder "Resources".

2. In the Solution Explorer window, right-click the **Resources** folder and select **Add | Existing Item**.

3. In the **Add Existing Item** dialog box, navigate to the **C1DataGrid_Demo\Resources** sample folder, select the **products.xml** file, and click **Add**. This file provides that data you'll use in the project.
4. Select the **products.xml** file in the Solution Explorer, and in the Properties window set its **Build Action** property to **Embedded Resource**.
5. In the Solution Explorer window, right-click the **MasterDetail** project and select **Add | Existing Item**.
6. In the **Add Existing Item** dialog box, navigate to the **C1DataGrid_Demo** sample folder, select the **Data.cs** file, and click **Add**. This file contains code to set up the data source.

What You've Accomplished

In this step you added an XML data source. In the next step, you'll set up the row details section and finalize the application.

Step 3 of 3: Setting up Row Details

In this step you'll finish setting up the row details section of the grid. You'll add a **RowDetailsTemplate** to set the appearance of the details row, and you'll add code to set up the details row behavior.

To set up row details, complete the following steps:

1. Add the following `<c1:C1DataGrid.RowDetailsTemplate>` between the `<c1:C1DataGrid></c1:C1DataGrid>` tags so that it appears similar to the following:
```
<c1:C1DataGrid x:Name="c1dg" CanUserAddRows="False" Margin="5">
    <c1:C1DataGrid.RowDetailsTemplate>
        <DataTemplate>
            <c1:C1DataGrid HeadersVisibility="Column" Margin="5"
CanUserAddRows="False"/>
        </DataTemplate>
    </c1:C1DataGrid.RowDetailsTemplate>
</c1:C1DataGrid>
```
   This template will customize the row details section display.
2. Add `LoadedRowDetailsPresenter="c1dg_LoadedRowDetailsPresenter" LoadingRow="c1dg_LoadingRow"` to the `<c1:C1DataGrid>` tag so that it appears similar to the following:
```
<c1:C1DataGrid x:Name="c1dg" CanUserAddRows="False"
LoadedRowDetailsPresenter="c1dg_LoadedRowDetailsPresenter"
LoadingRow="c1dg_LoadingRow">
```
   Later you'll add handlers for these events in code.
3. In the Solution Explorer, right-click the project and select **Add Reference**. In the **Add Reference** dialog box, locate **System.Xml.Linq** and **System.ComponentModel.DataAnnotations** and click **OK** to add the reference.
4. Right-click the page and select **View Code** in the context menu to open the Code Editor.
5. In the Code Editor, import the following namespaces:
   - C#
```
using System.Xml.Linq;
using C1.Silverlight.DataGrid;
using C1DataGrid_Demo;
```
6. Add code to the **Page** constructor to set the **ItemsSource** property:
   - C#
```
public MainPage()
{
    InitializeComponent();
    c1dg.ItemsSource = Data.GetSubCategories(null).Take(10);
}
```

7. Add code for the **c1dg_LoadedRowDetailsPresenter** event to the **MainPage** class:

   - C#

```
private void c1dg_LoadedRowDetailsPresenter(object sender,
C1.Silverlight.DataGrid.DataGridRowDetailsEventArgs e)
{
    if (e.Row.DetailsVisibility == Visibility.Visible)
    {
        C1.Silverlight.DataGrid.C1DataGrid detailGrid =
e.DetailsElement as C1.Silverlight.DataGrid.C1DataGrid;
        if (detailGrid.ItemsSource == null)
        {
            int subcategory = (e.Row.DataItem as
Subcategory).ProductSubcategoryID;
            detailGrid.ItemsSource = Data.GetProducts((product) =>
product.Element("ProductSubcategoryID") != null &&
product.Element("ProductSubcategoryID").Value != "" &&
int.Parse(product.Element("ProductSubcategoryID").Value) ==
subcategory).Take(10);
        }
    }
}
```

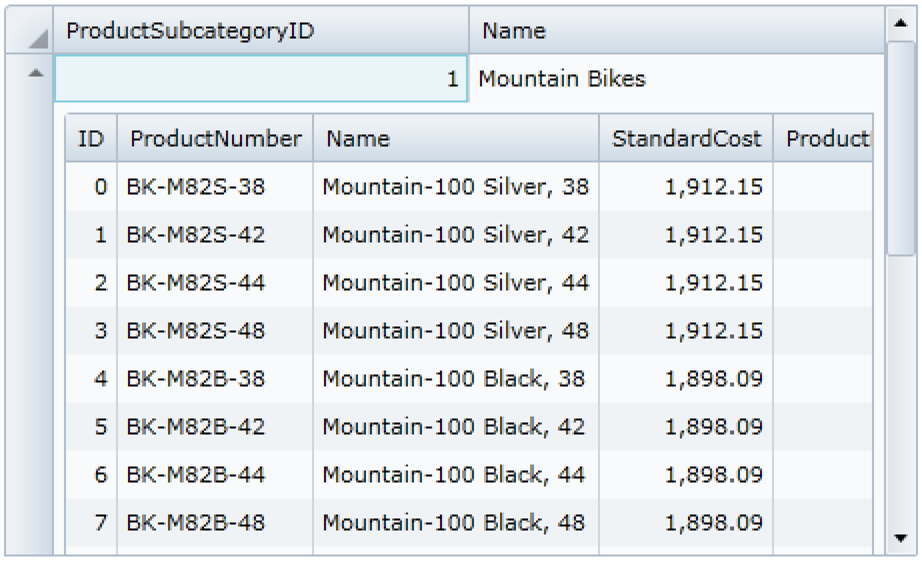
8. Add code for the **c1dg_LoadingRow** event to the **MainPage** class to set the row details visibility for the first row:

   - C#

```
private void c1dg_LoadingRow(object sender, DataGridRowEventArgs e)
{
    if (e.Row.Index == 0)
    {
        e.Row.DetailsVisibility = Visibility.Visible;
    }
}
```

✓ **What You've Accomplished**

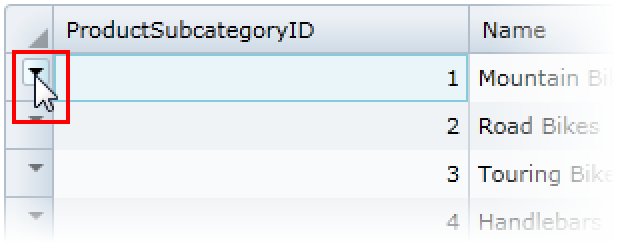
If you save and run your application you'll observe that the grid is now populated with data from the products.xml file, and that the first row's details section is visible:



ProductSubcategoryID	Name
1	Mountain Bikes

ID	ProductNumber	Name	StandardCost	Product
0	BK-M82S-38	Mountain-100 Silver, 38	1,912.15	
1	BK-M82S-42	Mountain-100 Silver, 42	1,912.15	
2	BK-M82S-44	Mountain-100 Silver, 44	1,912.15	
3	BK-M82S-48	Mountain-100 Silver, 48	1,912.15	
4	BK-M82B-38	Mountain-100 Black, 38	1,898.09	
5	BK-M82B-42	Mountain-100 Black, 42	1,898.09	
6	BK-M82B-44	Mountain-100 Black, 44	1,898.09	
7	BK-M82B-48	Mountain-100 Black, 48	1,898.09	

To collapse the row details section or expand another's row detail section, click the arrow icon in the row header of a row:



ProductSubcategoryID	Name
1	Mountain Bi
2	Road Bikes
3	Touring Bik
4	Handlebars

You've completed this tutorial and learned how to set up row details in the grid to display a master/detail grid view.

Localizing the Grid

Localizing **ComponentOne DataGrid for Silverlight** for various audiences is a fairly simple process as localization in Silverlight is based on the standard .NET localization. For more information about localization, see Localizing the Application (page 76). In this tutorial, you'll localize the visible grid strings in an existing application.

Step 1 of 3: Setting up the Localized Grid

In this step you'll create a Silverlight grid application using **ComponentOne DataGrid for Silverlight**. You'll create a new Silverlight project, add the **C1DataGrid** control to your application, and bind the grid.

Complete the following steps:

1. In Visual Studio, select **File | New | Project**.
2. In the **New Project** dialog box, select a language in the left pane, and in the templates list select **Silverlight Application**. Enter "C1DataGridLocalization" in the **Name** text box, and click **OK**. The **New Silverlight Application** dialog box will appear.
3. Click **OK** to close the **New Silverlight Application** dialog box and create your project.

4. In the `<UserControl>` tag, replace `Width="400"` (or `d:DesignWidth="400"`) with `Width="450"` to increase its size.

5. In the XAML window of the project, place the cursor between the `<Grid>` and `</Grid>` tags and click once.

6. Navigate to the Toolbox and double-click the **C1DataGrid** icon to add the grid control to **MainPage.xaml**. The XAML markup will now look similar to the following:

```
<UserControl xmlns:c1="http://schemas.componentone.com/winfx/2006/xaml"
x:Class="C1DataGridLocalization.MainPage"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml" Width="450"
Height="300">
    <Grid x:Name="LayoutRoot">
        <c1:C1DataGrid></c1:C1DataGrid>
    </Grid>
</UserControl>
```

7. If the `<c1:C1DataGrid>` tag includes existing content, delete it so it appears similar to the following:

```
<c1:C1DataGrid></c1:C1DataGrid>
```

8. Give your grid a name by adding `x:Name="c1dg"` to the `<c1:C1DataGrid>` tag so that it appears similar to the following:

```
<c1:C1DataGrid x:Name="c1dg">
```

   By giving the control a unique identifier, you'll be able to access the **C1DataGrid** control in code.

9. Add `CanUserGroup="True"` to the `<c1:C1DataGrid>` tag so that it appears similar to the following:

```
<c1:C1DataGrid x:Name="c1dg" CanUserGroup="True">
```

10. In the Solution Explorer, right-click the **C1DataGridLocalization** project and select **Build**.

11. In the Solution Explorer, right-click the **MainPage.xaml** file and click **View Code** in the context menu to open the Code Editor.

12. Add the following code to the project to create the **Data** class:

   - Visual Basic

```
Public Class Data
    Private _ProductName As String
    Public Property ProductName() As String
        Get
            Return _ProductName
        End Get
        Set(ByVal value As String)
            _ProductName = value
        End Set
    End Property
    Private _Description As String
    Public Property Description() As String
        Get
            Return _Description
        End Get
        Set(ByVal value As String)
            _Description = value
        End Set
    End Property
    Private _Quantity As Integer
    Public Property Quantity() As Integer
        Get
            Return _Quantity
        End Get
        Set(ByVal value As Integer)
            _Quantity = value
        End Set
    End Property
    Private _InStock As Boolean
    Public Property InStock() As Boolean
        Get
            Return _InStock
        End Get
        Set(ByVal value As Boolean)
            _InStock = value
        End Set
    End Property
End Class
```

- C#

```
public class Data
{
    public string ProductName { get; set; }
    public string Description { get; set; }
    public int Quantity { get; set; }
    public bool InStock { get; set; }
}
```

13. Add the following code to the **MainPage** constructor to populate the grid:

- Visual Basic

```
Public Sub New()
    InitializeComponent()
    ' Add data to a data source.

    Dim source As New List(Of Data)()
    Dim itemsCount As Integer = 25
    For i As Integer = 0 To itemsCount - 1
        source.Add(New Data With
            {
                .ProductName = "Name",
                .Description = "Description",
                .Quantity = i,
                .InStock = (i Mod 2 = 0)
            })
    Next
    ' Set the grid's ItemsSource property.
    c1dg.ItemsSource = source
End Sub
```

- C#

```
public MainPage()
{
    InitializeComponent();

    // Add data to a data source.
    List<Data> source = new List<Data>();
    int itemsCount = 25;

    for (int i = 0; i < itemsCount; i++)
    {
        source.Add(new Data()
        {
            ProductName = "Name",
            Description = "Description",
            Quantity = i,
            InStock = (i % 2 == 0)
        });
    }

    // Set the grid's ItemsSource property.
    c1dg.ItemsSource = source;
}
```

✓ **What You've Accomplished**

In this step you created a new Silverlight application, added a **C1DataGrid** control, and bound the control to a data source. In the next step, you'll add a resource file to localize the grid.

Step 2 of 3: Adding a Resource File

In this step, you'll begin by adding a resource file to your application. Note that if you choose, you can add multiple resources files to your project.

1. In the Solution Explorer, right-click the **C1DataGridLocalization** project and choose **Add | New Folder**.
2. Name the folder you just created "Resources".
3. Right-click the **Resources** folder, and in the context menu select **Add | New Item**.
4. In the **Add New Item** dialog box, select **Resources File** in the templates pane, name the file "C1.Silverlight.DataGrid.resx", and click **Add** to add the file to your project.
5. If the resource file did not automatically open, double-click the file name in the Solution Explorer.
6. In the C1.Silverlight.DataGrid.es.resx file, add the following Names and Values:

Name	**Value**
AddNewRow	Click here to add a new row
CheckBoxFilter_Checked	Checked:
ComboBoxFilter_SelectAll	Select All
DateTimeFilter_End	End
DateTimeFilter_Start	Start
EmptyGroupPanel	Drag a column here to group by that column
Filter_Clear	Clear
Filter_Filter	Filter
NumericFilter_And	And
NumericFilter_Equals	Equals
NumericFilter_GraterOrEquals	Greater/Equals
NumericFilter_Greater	Greater
NumericFilter_Less	Less
NumericFilter_LessOrEquals	Less/Equals
NumericFilter_NotEquals	Not Equals
NumericFilter_Or	Or

TextFilter_Contains	Contains
TextFilter_StartsWith	Starts With
TextFilter_Equals	Equals
TextFilter_NotEquals	Not Equals

7. Save and close the resource file.
8. Right-click the **Resources** folder, and in the context menu select **Add | New Item**.
9. In the **Add New Item** dialog box, select **Resources File** in the templates pane, name the file "C1.Silverlight.DataGrid.es.resx", and click **Add** to add the file to your project.

   This file will localize the application to Spanish. For information on file naming, see Adding Resource Files (page 76).
10. If the resource file did not automatically open, double-click the file name in the Solution Explorer.
11. In the C1.Silverlight.DataGrid.es.resx file, add the following Names and Values to add Spanish localization:

Name	**Value**
AddNewRow	Cliquee aquí para agregar un nuevo renglón
CheckBoxFilter_Checked	Seleccionado:
ComboBoxFilter_SelectAll	Seleccionar todo
DateTimeFilter_End	Fin
DateTimeFilter_Start	Inicio
EmptyGroupPanel	Arrastre una columna aquí para agrupar
Filter_Clear	Borrar
Filter_Filter	Filtrar
NumericFilter_And	Y
NumericFilter_Equals	Igual
NumericFilter_GraterOrEquals	Mayor o igual
NumericFilter_Greater	Mayor
NumericFilter_Less	Menor
NumericFilter_LessOrEquals	Menor o igual
NumericFilter_NotEquals	Diferente
NumericFilter_Or	O
TextFilter_Contains	Contiene
TextFilter_StartsWith	Empieza con
TextFilter_Equals	Igual
TextFilter_NotEquals	Diferente

12. Save and close the resource file.

What You've Accomplished

In this step you added a new resource file to your application. In the next step you'll add the file's culture to the project's supported cultures, and then set that culture to be the current culture.

Step 3 of 3: Setting the Culture

Once you've created resource files for your application, you will need to set the supported cultures for your project and explicitly set the current culture of the project. To do so, complete the following steps:

1. In the Solution Explorer, right-click the **C1DataGridLocalization** project and select **Unload Project**. Click **Yes** if Visual Studio asks you to save the project.

   The project will appear grayed out and unavailable.

2. Right-click the project again, and select the **Edit C1DataGridLocalization.csproj** option.

   In the .csproj file, locate the `<SupportedCultures></SupportedCultures>` tags. Add "es" in between the tags, so they appear similar to the following:
   ```
   <SupportedCultures>es</SupportedCultures>
   ```

3. Save and close the .csproj file.

4. In the Solution Explorer, right-click your project and choose **Reload Project** from the context menu.

   The project will be reloaded and will now support the specified cultures.

5. In the Solution Explorer, right-click the **MainPage.xaml** file and click **View Code** in the context menu to open the Code Editor.

6. Add the following using statements to the top of the file:

   - Visual Basic
     ```
     Imports System.Globalization
     Imports System.Threading
     ```

   - C#
     ```
     using System.Globalization;
     using System.Threading;
     ```

7. Add the following code to the **MainPage** constructor above the **InitializeComponent()** call to set the **CurrentUICulture** property:

   - Visual Basic
     ```
     Thread.CurrentThread.CurrentUICulture = New CultureInfo("es")
     ```

   - C#
     ```
     Thread.CurrentThread.CurrentUICulture = new CultureInfo("es");
     ```

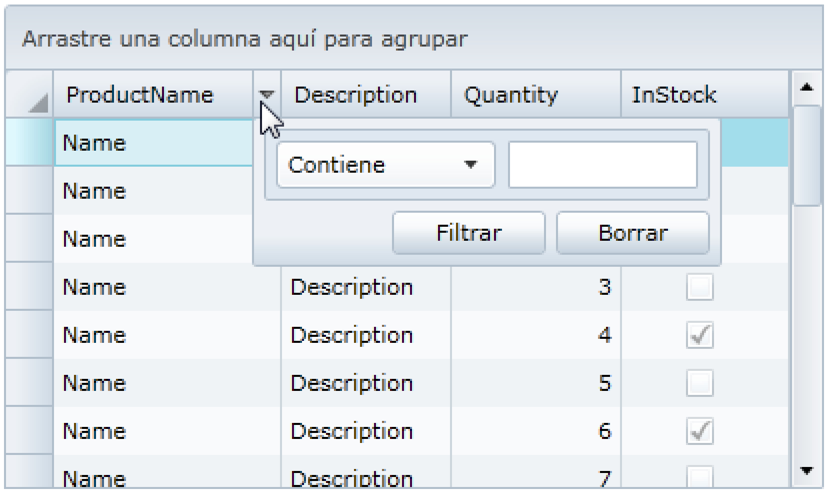
   It should now look similar to the following:

   - Visual Basic
     ```
     Public Sub New()
         ' Set the culture.
         Thread.CurrentThread.CurrentUICulture = New CultureInfo("es")
         InitializeComponent()
         ' Add data to a data source.
         Dim source As New List(Of Data)()
         Dim itemsCount As Integer = 25
         For i As Integer = 0 To itemsCount - 1
             source.Add(New Data())
         Next
         ' Set the grid's ItemsSource property.
         c1dg.ItemsSource = source
     End Sub
     ```

- C#

```
public MainPage()
{
    // Set the culture.
    Thread.CurrentThread.CurrentUICulture = new CultureInfo("es");
    InitializeComponent();
    // Add data to a data source.
    List<Data> source = new List<Data>();
    int itemsCount = 25;
    for (int i = 0; i < itemsCount; i++)
    {
        source.Add(new Data()
        {
            ProductName = "Name",
            Description = "Description",
            Quantity = i,
            InStock = (i % 2 == 0)
        });
    }
    // Set the grid's ItemsSource property.
    c1dg.ItemsSource = source;
}
```

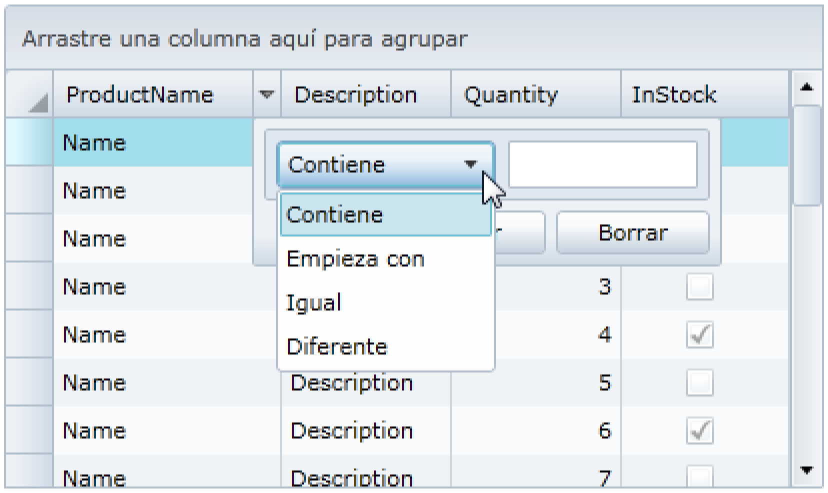
8. Save and run your application.
9. To observe some of the localized language strings, select the drop-down filter icon in the grid:



Arrastre una columna aquí para agrupar

ProductName	Description	Quantity	InStock
Name			
Name			
Name			
Name	Description	3	☐
Name	Description	4	✓
Name	Description	5	☐
Name	Description	6	✓
Name	Description	7	☐

10. Click the drop-down arrow in the filter box to view additional strings:



What You've Accomplished

In this step you added the file's culture to the project's supported cultures and set that culture to be the current culture. In this tutorial you've learned how to localize an application. You created a resource file, set the project's supported culture, and explicitly set the current culture in code.

Binding the Grid to a WCF RIA Services Data Source

The following tutorial will walk you through the process of binding the **C1DataGrid** control to a WCF RIA Services data source. For more information, see the WCF RIA Services Data Binding (page 58) topic. Note that this example will use files included in the **C1DataGrid_Ria2010** sample installed with **ComponentOne Studio for Silverlight**.

Step 1 of 3: Creating the Application and Adding the Data Source

In this step you'll create a new Silverlight project with WCF RIA services enabled, add a data source, and set up the client-side project. Complete the following steps:

1. In Visual Studio 2010, select **File | New | Project**.
2. In the **New Project** dialog box, choose **Visual C#** in the left pane, and in the templates list select **Silverlight Application**. Enter "C1DataGridRIA" in the **Name** text box, and click **OK**. The **New Silverlight Application** dialog box will appear.
3. In the **New Silverlight Application** dialog box, check the **Enable WCF RIA Services** check box and click **OK** to close the **New Silverlight Application** dialog box and create your project.
4. In the Solution Explorer, right-click the **C1DataGridRIA.Web** project and choose **Add | New Folder**. Rename the folder "App_Data".
5. Right-click the **App_Data** folder and select **Add | Existing Item**.
6. In the **Add Existing Item** dialog box, navigate to where the ComponentOne samples are installed, by default in the **Documents** or **My Documents** folder, and navigate to the **ComponentOne Samples\Studio for Silverlight 4.0\C1.Silverlight.DataGrid\C1DataGrid_Ria\C1DataGrid_Ria2010Web\App_Data** folder. Choose the **NORTHWND.MDF** file and click the **Add** button.

   The database will be added to your project. Note that this is the standard Microsoft Northwind database.
7. In the Solution Explorer, right-click the **C1DataGridRIA.Web** project and choose **Add | New Item**.
8. In the Add New Item dialog box choose **Data** in the left list and choose **ADO.NET Entity Data Model** from the list of data templates. Name the file "NorthwindModel" and click **Add** to add the file to your project.

9. The **Entity Data Model Wizard** should appear. Choose the **Generate from database** option and click **Next**.
10. In the **Choose Your Data Connection** screen, confirm that the **NORTHWND.MDF** file is selected. If it is not selected, choose **New Connection** and locate the file. Save the connection string with the default name, "NORTHWNDEntities", and click **Next**.
11. In the **Choose Your Database Objects** screen, select the **Tables** check box to choose the **Products** table. Click **Finish**.
12. Choose **Build | Rebuild Solution** to build the entire solution and make sure the autogenerated RIA Services files get created.
13. In the Solution Explorer, right-click the **C1DataGridRIA.Web** project and choose **Add | New Item**.
14. In the **Add New Item** dialog box choose **Web** in the left list and choose **Domain Service Class** from the list of code templates. Name the file "NorthwindService" and click **Add** to add the file to your project. The **Add New Domain Service Class** dialog box will appear.
15. Iin the **Add New Domain Service Class** dialog box, select **NorthwindEntities**as DataContext item and select the **Enable client access** check box. Check the **Product** entity and **Enable editing** check boxes and click **OK**.
16. Save the project and choose **Build | Rebuild Solution** to ensure everything is working correctly.

What You've Accomplished

In this step you added a new RIA data source to your application. In the next step you'll add the **C1DataGrid** control to the application.

Step 2 of 3: Adding the C1DataGrid control

In the previous step you created a new Silverlight application with WCF RIA services enabled and added a new data source. In this step you'll set up your application and add the **C1DataGrid** control to the application. Complete the following steps:

1. In the Solution Explorer, right click the **C1DataGridRIA** project and choose **Add Reference**. The **Add Reference** dialog box will appear.
2. In the **Add Reference** dialog box, select the following assemblies and then click **OK**:
   - System.Windows.Controls.Data
   - System.Windows.Controls.DomainServices
   - C1.Silverlight
   - C1.Silverlight.DataGrid
   - C1.Silverlight.DataGrid.Ria

   This will add references to the project for the selected assemblies.
3. In the Solution Explorer, double-click the **MainPage.xaml** file to open it.
4. In the XAML window of the project, update the **UserControl** tag so it appears similar to the following:
   - XAML

```
<UserControl x:Class="C1DataGridRIA.MainPage"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc="http://schemas.openxmlformats.org/markup-
compatibility/2006"
```

```
    xmlns:data="clr-
namespace:System.Windows.Controls;assembly=System.Windows.Controls.Data
"
    xmlns:ria="clr-
namespace:System.Windows.Controls;assembly=System.Windows.Controls.Doma
inServices"
    xmlns:c1data="clr-
namespace:C1.Silverlight.DataGrid;assembly=C1.Silverlight.DataGrid"
    xmlns:adapter="clr-
namespace:C1.Silverlight.DataGrid.Ria;assembly=C1.Silverlight.DataGrid.
Ria"
    xmlns:local="clr-namespace:C1DataGridRIA.Web"
    mc:Ignorable="d" d:DesignWidth="640" d:DesignHeight="480">
```

This markup will add references to the assemblies you added, and resize the **UserControl**.

5. Add the following markup just after the **Grid** tag to create a row definition:

- XAML

```
<Grid.RowDefinitions>
    <RowDefinition Height="Auto"/>
    <RowDefinition Height="*"/>
</Grid.RowDefinitions>
```

This markup will set the layout of the page.

6. Add the following markup within the **Grid** tag and just under the row definitions to create a **C1RiaAdaptor**:

- XAML

```
<!-- RIA Data Source -->

<adapter:C1RiaAdapter x:Name="_adapter" DataGrid="{Binding
ElementName=_dataGrid}">

    <ria:DomainDataSource x:Name="_myDataSource"
QueryName="GetProducts" PageSize="8">

        <ria:DomainDataSource.DomainContext>

            <local:NorthwindContext/>

        </ria:DomainDataSource.DomainContext>

        <ria:DomainDataSource.GroupDescriptors>

            <ria:GroupDescriptor PropertyPath="CategoryID"/>

            <ria:GroupDescriptor PropertyPath="Discontinued"/>

        </ria:DomainDataSource.GroupDescriptors>

        <ria:DomainDataSource.SortDescriptors>

            <ria:SortDescriptor PropertyPath="ProductName"
Direction="Descending"/>

        </ria:DomainDataSource.SortDescriptors>

        <ria:DomainDataSource.FilterDescriptors>

            <ria:FilterDescriptor PropertyPath="UnitPrice"
Operator="IsGreaterThanOrEqualTo" Value="18"/>

            <ria:FilterDescriptor PropertyPath="ProductName"
Operator="Contains" Value="C"/>
```

```
        </ria:DomainDataSource.FilterDescriptors>
    </ria:DomainDataSource>
</adapter:C1RiaAdapter>
```

This markup will add the RIA data source.

7. Add the following markup within the **Grid** tag and under the **C1RiaAdaptor** tag to add a header to the page:

   - XAML

```
<!-- Header -->
<Border Grid.Row="0" Height="40" Background="LightBlue">
    <TextBlock Text="CollectionView adapter for C1DataGrid: RIA
Services"
       Margin="10 0 0 0" FontSize="15" FontWeight="Bold"
VerticalAlignment="Center"/>
</Border>
```

8. Add the following markup within the **Grid** tag and under the Header to add a layout **Grid** to the page:

   - XAML

```
<!-- Content -->
<Grid Grid.Row="1" Margin="20">
    <Grid.RowDefinitions>
        <RowDefinition Height="Auto"/>
        <RowDefinition Height="Auto"/>
        <RowDefinition Height="Auto"/>
        <RowDefinition Height="Auto"/>
        <RowDefinition Height="*"/>
    </Grid.RowDefinitions>
</Grid>
```

   You will add the **C1DataGrid** control within this layout grid.

9. Add the following markup within within the content layout **Grid** you just added (just above the `</Grid>` tag) to add a standard **DataPager** control to the page:

   - XAML

```
<!-- DataPager -->
    <data:DataPager x:Name="_dataPager" Source="{Binding Data,
ElementName=_myDataSource}" BorderThickness="0" Background="White"/>
```

10. Add the following markup within within the content layout **Grid** and just after the **DataPager** to add a **C1DataGrid** control to the page:

   - XAML

```
<!-- C1DataGrid -->
<c1data:C1DataGrid x:Name="_dataGrid" CanUserGroup="True"
AutoGenerateColumns="False" Grid.Row="1"
```

```
        CanUserAddRows="True" CanUserEditRows="True"
CanUserRemoveRows="True"
        ItemsSource="{Binding Data, ElementName=_adapter}"
        BeginningRowEdit="_dataGrid_BeginningRowEdit"
CommittingRowEdit="_dataGrid_CommittingRowEdit"
        CancelingRowEdit="_dataGrid_CancelingRowEdit"
RowsDeleted="_dataGrid_RowsDeleted" >
    <c1data:C1DataGrid.Columns>
        <c1data:DataGridNumericColumn Binding="{Binding CategoryID,
Mode=TwoWay}" SortMemberPath="CategoryID" FilterMemberPath="CategoryID"
Header="CategoryID"/>
        <c1data:DataGridCheckBoxColumn Binding="{Binding Discontinued,
Mode=TwoWay}" SortMemberPath="Discontinued"
FilterMemberPath="Discontinued" Header="Discontinued"/>
        <c1data:DataGridTextColumn Binding="{Binding ProductName,
Mode=TwoWay}" SortMemberPath="ProductName"
FilterMemberPath="ProductName" Header="ProductName"/>
        <c1data:DataGridTextColumn Binding="{Binding QuantityPerUnit,
Mode=TwoWay}" SortMemberPath="QuantityPerUnit"
FilterMemberPath="QuantityPerUnit" Header="QtyPerUnit"/>
        <c1data:DataGridNumericColumn Binding="{Binding UnitPrice,
Mode=TwoWay}" SortMemberPath="UnitPrice" FilterMemberPath="UnitPrice"
Header="UnitPrice"/>
    </c1data:C1DataGrid.Columns>
</c1data:C1DataGrid>
```

This **C1DataGrid** control is bound to the database added earlier and includes defined and bound columns.

11. Add the following markup within within the content layout **Grid** and just after the **C1DataGrid** to add a text box and two buttons to the page:

- XAML

```
<!-- Change Text -->
<TextBox x:Name="_changeText" Margin="0 4 0 0" Grid.Row="2"/>


<!-- Reject Button -->
<Button x:Name="_rejectButton" Content="Reject Changes"
IsEnabled="False" Click="_rejectButton_Click" Width="120"
HorizontalAlignment="Right" Margin="0 4 130 0" Grid.Row="3"/>


<!-- Submit Button -->
<Button x:Name="_submitButton" Content="Submit Changes"
IsEnabled="False" Click="_submitButton_Click" Width="120"
HorizontalAlignment="Right" Margin="0 4 0 0" Grid.Row="3"/>
```

At run time, the text box will display the location of any changes made to the grid and the buttons will allow you to reject or apply any changes made to the grid at run time. In the next step you'll add code to implement the XAML you added to the application.

12. Right-click the **MainPage.xaml** page and choose **View Code** to open the **MainPage.xaml.cs** (or **MainPage.xaml.vb**) page in the Code Editor.

13. Add the imports statement to the top of the page:

    - Visual Basic

```
Imports C1.Silverlight.DataGrid
Imports System.ServiceModel.DomainServices.Client
```

    - C#

```
using C1.Silverlight.DataGrid;
using System.ServiceModel.DomainServices.Client;
```

14. Add the following code within the **MainPage** class to implement the controls that were added in XAML:

    - Visual Basic

```
Private Sub _submitButton_Click(sender As Object, e As RoutedEventArgs)
   ' Submit changes to the server
   _dataGrid.IsLoading = True
   _myDataSource.DomainContext.SubmitChanges(AddressOf
OnSubmitCompleted, Nothing)
End Sub
Private Sub _rejectButton_Click(sender As Object, e As RoutedEventArgs)
   ' Reject changes
   _myDataSource.DomainContext.RejectChanges()
   CheckChanges()
   _dataGrid.Reload(False)
End Sub
' Disable submit/reject buttons when there are pending changes in the
row
Private Sub _dataGrid_BeginningRowEdit(sender As Object, e As
DataGridEditingRowEventArgs)
   _submitButton.IsEnabled = False
   _rejectButton.IsEnabled = False
End Sub
' Enable/disable submit/reject buttons after pending changes are
committed
Private Sub _dataGrid_CommittingRowEdit(sender As Object, e As
DataGridEditingRowEventArgs)
   CheckChanges()
End Sub
' Enable/disable submit/reject buttons after pending changes are
canceled
Private Sub _dataGrid_CancelingRowEdit(sender As Object, e As
DataGridEditingRowEventArgs)
   CheckChanges()
End Sub
' Enable/disable submit/reject buttons after rows deleted
Private Sub _dataGrid_RowsDeleted(sender As Object, e As
DataGridRowsDeletedEventArgs)
   CheckChanges()
End Sub
' Check the pending changes to submit/reject and enable/disable buttons
according to this.
Private Sub CheckChanges()
   Dim changeSet As EntityChangeSet =
_myDataSource.DomainContext.EntityContainer.GetChanges()
   _changeText.Text = changeSet.ToString()
   Dim hasChanges As Boolean = _myDataSource.HasChanges
   _submitButton.IsEnabled = hasChanges
   _rejectButton.IsEnabled = hasChanges
End Sub
' Check for errors when submitting changes to the server
Private Sub OnSubmitCompleted(so As SubmitOperation)
   _dataGrid.IsLoading = False
   If so.HasError Then
         MessageBox.Show(String.Format("Submit Failed: {0}",
so.[Error].Message))
         so.MarkErrorAsHandled()
   End If
   CheckChanges()
End Sub
```

- C#

```
private void _submitButton_Click(object sender, RoutedEventArgs e)
{
    // Submit changes to the server
    _dataGrid.IsLoading = true;
    _myDataSource.DomainContext.SubmitChanges(OnSubmitCompleted, null);
}
private void _rejectButton_Click(object sender, RoutedEventArgs e)
{
    // Reject changes
    _myDataSource.DomainContext.RejectChanges();
    CheckChanges();
    _dataGrid.Reload(false);
}
// Disable submit/reject buttons when there are pending changes in the
row
private void _dataGrid_BeginningRowEdit(object sender,
DataGridEditingRowEventArgs e)
{
    _submitButton.IsEnabled = false;
    _rejectButton.IsEnabled = false;
}
// Enable/disable submit/reject buttons after pending changes are
committed
private void _dataGrid_CommittingRowEdit(object sender,
DataGridEditingRowEventArgs e)
{
    CheckChanges();
}
// Enable/disable submit/reject buttons after pending changes are
canceled
private void _dataGrid_CancelingRowEdit(object sender,
DataGridEditingRowEventArgs e)
{
    CheckChanges();
}
// Enable/disable submit/reject buttons after rows deleted
private void _dataGrid_RowsDeleted(object sender,
DataGridRowsDeletedEventArgs e)
{
    CheckChanges();
}
```

```
// Check the pending changes to submit/reject and enable/disable
buttons according to this.
private void CheckChanges()
{
    EntityChangeSet changeSet =
_myDataSource.DomainContext.EntityContainer.GetChanges();
    _changeText.Text = changeSet.ToString();

    bool hasChanges = _myDataSource.HasChanges;
    _submitButton.IsEnabled = hasChanges;
    _rejectButton.IsEnabled = hasChanges;
}
// Check for errors when submitting changes to the server
private void OnSubmitCompleted(SubmitOperation so)
{
    _dataGrid.IsLoading = false;
    if (so.HasError)
    {
        MessageBox.Show(string.Format("Submit Failed: {0}",
so.Error.Message));
        so.MarkErrorAsHandled();
    }
    CheckChanges();
}
```

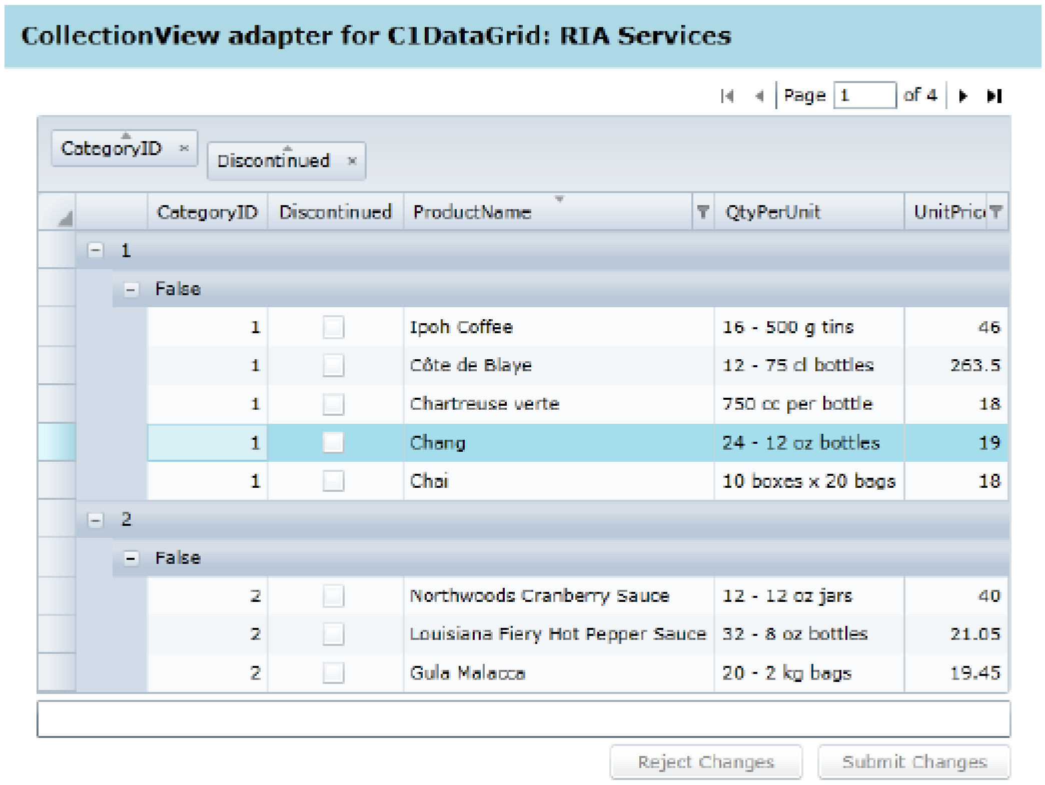
What You've Accomplished

You learned how to bind the **C1DataGrid** control to an RIA Services data source. You created a Silverlight application, added the data source, and added and implemented the **C1DataGrid** control. In the next step you'll run the application, to view its run time interactions.

Step 3 of 3: Running the Application

In the previous steps you created a new Silverlight application with WCF RIA services enabled, added a new data source, and added the **C1DataGrid** control to the application. In this step you'll run the application, to view its run time interactions. Complete the following steps:

1. Save the project and choose **Debug | Start Debugging** to run the application. It will appear similar to the following image:


CollectionView adapter for C1DataGrid: RIA Services

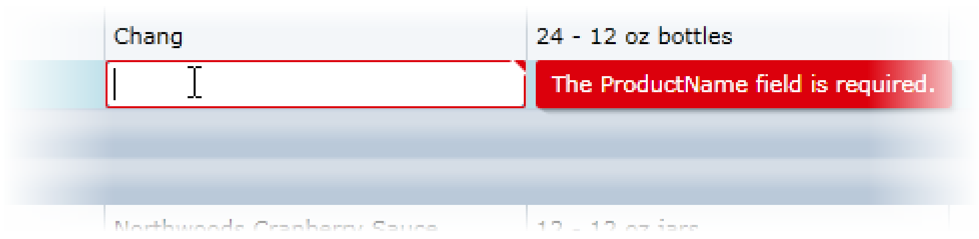
Page 1 of 4

CategoryID × Discontinued ×

CategoryID	Discontinued	ProductName	QtyPerUnit	UnitPrice
1				
False				
1	☐	Ipoh Coffee	16 - 500 g tins	46
1	☐	Côte de Blaye	12 - 75 cl bottles	263.5
1	☐	Chartreuse verte	750 cc per bottle	18
1	☐	Chang	24 - 12 oz bottles	19
1	☐	Chai	10 boxes x 20 bags	18
2				
False				
2	☐	Northwoods Cranberry Sauce	12 - 12 oz jars	40
2	☐	Louisiana Fiery Hot Pepper Sauce	32 - 8 oz bottles	21.05
2	☐	Gula Malacca	20 - 2 kg bags	19.45

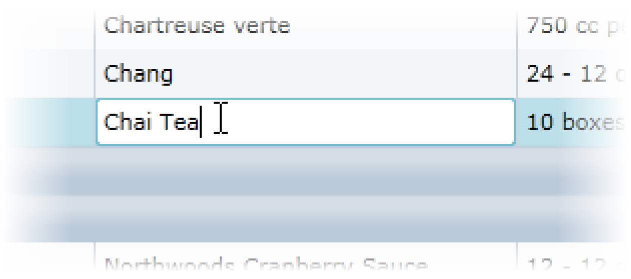
Reject Changes Submit Changes

2. At run time, click on a cell in the ProductName column and delete the text from a cell. Notice that validation text appears:


Chang | 24 - 12 oz bottles

The ProductName field is required.

3. Enter text in the cell in the ProductName column you deleted:

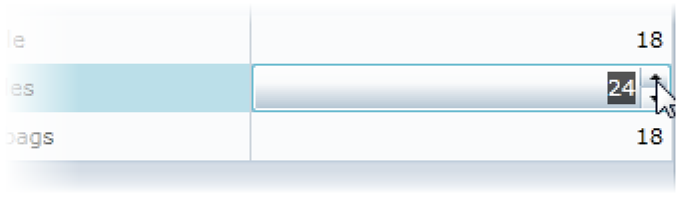

Chartreuse verte | 750 cc p

Chang | 24 - 12

Chai Tea | 10 boxe

Click away drom the cell you edited and notice that the box under the grid notes that one cell in the grid has been modified and the buttons below the grid are now active.

4. Click the **Reject Changes** button to discard the changes you made.
5. Click an item in the UnitPrice column and use the up and down arrows to change the value of the cell:



6. Click away from the cell and click the **Submit Changes** button to save your changes to the data.

What You've Accomplished

In this tutorial you learned how to bind the **C1DataGrid** control to an RIA Services data source. You created a Silverlight application, added the data source, and added and implemented the **C1DataGrid** control.

Implementing Stealth Paging

With paging you can only load the necessary data to fit one page. See Paging Grid Data (page 80) for details. Stealth paging is a little different; you can achieve paging functionality with a scrollbar. As the user scrolls down the grid, more data is fetched as needed, just like with paging. **C1DataGrid** supports server-side sorting and filtering so you can still achieve these functionalities without sacrificing performance. In this tutorial you'll create a Silverlight application with a **C1DataGrid** control that implements stealth paging functionality.

Step 1 of 3: Creating the User Interface

In this step you'll begin in Visual Studio to create a Silverlight grid application. You'll then continue by creating and customizing the application's user interface (UI) and adding the **C1DataGrid** control to your project.

To set up your project, complete the following steps:

1. In Visual Studio, select **File | New | Project**.
2. In the **New Project** dialog box, select a language in the left pane and in the templates list select **Silverlight Application**. Enter a **Name** for your project "StealthPaging", and click **OK**. The **New Silverlight Application** dialog box will appear.
3. Click **OK** to accept the default settings, close the **New Silverlight Application** dialog box, and create your project.
4. Navigate to the Solution Explorer, right-click the **StealthPaging** project, and select **Add Reference** from the context menu.
5. In the **Add Reference** dialog box locate the **System.Runtime.Serialization** assembly and click the **OK** button to add a reference to your project. The dialog box will close and the reference will be added.
6. If the **MainPage.xaml** file is not currently open, navigate to the Solution Explorer and double-click on the **MainPage.xaml** item.
7. In the XAML view, place the cursor just after the `<Grid x:Name="LayoutRoot" Background="White">` tag and add the following markup:

```
<!-- Grid Layout-->
<Grid.RowDefinitions>
    <RowDefinition Height="*" />
    <RowDefinition Height="Auto" />
</Grid.RowDefinitions>
```

This row definition will define the layout of your application.

8. In the XAML window of the project, place the cursor just above the `</Grid>` tag and click once.
9. Navigate to the Toolbox and double-click the **C1DataGrid** icon to add the grid control to **MainPage.xaml**. The XAML markup will now look similar to the following:

   ```
   <UserControl x:Class="StealthPaging.MainPage"
       xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
       xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
       xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
       xmlns:mc="http://schemas.openxmlformats.org/markup-
   compatibility/2006"
       mc:Ignorable="d"
       d:DesignHeight="300" d:DesignWidth="400"
   xmlns:c1="http://schemas.componentone.com/winfx/2006/xaml">
       <Grid x:Name="LayoutRoot" Background="White">
           <!-- Grid Layout-->
           <Grid.RowDefinitions>
               <RowDefinition Height="*" />
               <RowDefinition Height="Auto" />
           </Grid.RowDefinitions>
           <c1:C1DataGrid />
       </Grid>
   </UserControl>
   ```

   Note that the **C1.Silverlight.DataGrid** namespace and `<c1:C1DataGrid />` tag has been added to the project.
10. If the `<c1:C1DataGrid>` tag includes existing content, delete it so it appears similar to the following:

    ```
    <c1:C1DataGrid />
    ```
11. Give your grid a name by adding `x:Name="peopleDataGrid"` to the `<c1:C1DataGrid>` tag so that it appears similar to the following:

    ```
    <c1:C1DataGrid x:Name="peopleDataGrid" />
    ```

    By giving the control a unique identifier, you'll be able to access the **C1DataGrid** control in code.
12. Customize your grid by adding `AutoGenerateColumns="True" CanUserAddRows="False"` to the `<c1:C1DataGrid>` tag so that it appears similar to the following:

    ```
    <c1:C1DataGrid x:Name="peopleDataGrid" AutoGenerateColumns="True"
    CanUserAddRows="False" />
    ```

    This markup will set the grid to generate columns automatically and will disable adding new rows.
13. Add the following markup just after the `</c1:C1DataGrid>` tag:

    ```
    <TextBlock x:Name="txtStatus" Grid.Row="1" Text="Ready."  Margin="0,5,0,0"
    />
    ```

    This **TextBlock** will be used to display status information text.

✓ **What You've Accomplished**

If you run your application you'll observe that your page includes a grid and text below the grid. You've successfully created a basic grid application, but the grid is blank and contains no data. In the next steps you'll bind the grid to a data source and add stealth paging in code.

Step 2 of 3: Adding a Web Service

In this step you'll add a data source to your project, and begin the process of binding the grid.

To set up your project, complete the following steps:

1. Navigate to the Solution Explorer, right-click the **StealthPaging.Web** project, and select **Add Reference** from the context menu.
2. In the **Add Reference** dialog box locate the **System.Runtime.Serialization** assembly and click the **OK** button to add a reference to your project. The dialog box will close and the reference will be added.
3. In the Solution Explorer right-click the **StealthPaging.Web** project, and select **Add | New Item**.
4. In the left pane of the **Add New Item** dialog box, select the **Web** item.
5. In the templates list, select **Web Service**, name the Web Service "DataWebService.asmx", and click the **Add** button. Note that the Web Service file will be added to your project and automatically opened.
6. In the **DataWebService.asmx** file, add the following using statements at the top of the file:
   - Visual Basic
     ```
     Imports System.Runtime.Serialization
     ```
   - C#
     ```
     using System.Runtime.Serialization;
     ```
7. In the **DataWebService.asmx** file, replace the code in the **StealthPaging.Web** namespace with the following::
   - Visual Basic
     ```
     ' To allow this Web Service to be called from script, using ASP.NET
     AJAX, uncomment the following line.
     ' <System.Web.Script.Services.ScriptService()> _
     <System.Web.Services.WebService(Namespace:="http://tempuri.org/")> _
     <System.Web.Services.WebServiceBinding(ConformsTo:=WsiProfiles.BasicPro
     file1_1)> _
     <ToolboxItem(False)> _
     Public Class DataWebService
         Inherits System.Web.Services.WebService

         <WebMethod()> _
              Public Function GetData(startRow As Integer, endRow As
     Integer) As List(Of ServerPerson)
                    Dim personList As New List(Of ServerPerson)()
                    For i As Integer = startRow To endRow - 1
                          personList.Add(New ServerPerson() With { _
                                .FirstName = String.Format("First Name {0}",
     i), _
                                .LastName = String.Format("Last Name {0}",
     i), _
                                .Age = i, _
                                .City = String.Format("City {0}", i) _
                          })
                    Next

                    Return personList
              End Function
        End Class

        <DataContract> _
        Public Class ServerPerson
              Private _firstName As String
              <DataMember> _
              Public Property FirstName() As String
     ```

```
                Get
                    Return _firstName
                End Get
                Set
                    _firstName = value
                End Set
            End Property

            Private _lastName As String
            <DataMember> _
            Public Property LastName() As String
                Get
                    Return _lastName
                End Get
                Set
                    _lastName = value
                End Set
            End Property

            Private _age As Integer
            <DataMember> _
            Public Property Age() As Integer
                Get
                    Return _age
                End Get
                Set
                    _age = value
                End Set
            End Property

            Private _city As String
            <DataMember> _
            Public Property City() As String
                Get
                    Return _city
                End Get
                Set
                    _city = value
                End Set
            End Property
        End Class
```

- C#

```
namespace StealthPaging.Web
{
    /// <summary>
    /// Summary description for DataWebService
    /// </summary>
    [WebService(Namespace = "http://tempuri.org/")]
    [WebServiceBinding(ConformsTo = WsiProfiles.BasicProfile1_1)]
    [System.ComponentModel.ToolboxItem(false)]
    // To allow this Web Service to be called from script, using
ASP.NET AJAX, uncomment the following line.
    // [System.Web.Script.Services.ScriptService]
    public class DataWebService : System.Web.Services.WebService
    {
        [WebMethod]
        public List<ServerPerson> GetData(int startRow, int endRow)
        {
            List<ServerPerson> personList = new List<ServerPerson>();
            for (int i = startRow; i < endRow; i++)
            {
                personList.Add(new ServerPerson()
                {
                    FirstName = string.Format("First Name {0}", i),
                    LastName = string.Format("Last Name {0}", i),
                    Age = i,
                    City = string.Format("City {0}", i)
                });
            }

            return personList;
        }
    }

    [DataContract]
    public class ServerPerson
    {
        private string _firstName;
        [DataMember]
        public string FirstName
        {
            get { return _firstName; }
            set { _firstName = value; }
        }

        private string _lastName;
        [DataMember]
        public string LastName
        {
            get { return _lastName; }
            set { _lastName = value; }
        }

        private int _age;
        [DataMember]
        public int Age
        {
            get { return _age; }
            set { _age = value; }
        }

        private string _city;
        [DataMember]
        public string City
        {
            get { return _city; }
            set { _city = value; }
        }
    }
}
```

This code will create a new list that will be used to populate the C1DataGrid control.

8. Save your application, right-click the **StealthPaging.Web** project, and select **Build** from the context menu. Note that you'll now be done with the **StealthPaging.Web** project and will return to working with the **StealthPaging** project.

What You've Accomplished

In this step you've added a data source to your project and created a Web Service. In the next step you'll finish connecting the Web Service to your project and you'll run your application.

Step 3 of 3: Connecting the Web Service and Adding Stealth Paging

In the previous step you created a Web Service and added a data source to your project. In this step you'll continue by linking the Web Service to your application.

To set up your project, complete the following steps:

1. Return to the **MainPage.xaml** file.
2. In the Solution Explorer, right-click the project name and select **Add Service Reference** from the context menu.
3. In the **Add Service Reference** dialog box click the **Discover** button. The DataWebService.asmx file will appear in the list of Services.
4. In the **Namespace** text box, change the default value to "DataService" and click the **OK** button to save your settings and close the dialog box.
5. Customize your grid by adding `LoadedRowPresenter="peopleDataGrid_LoadedRowPresenter"` to the `<c1:C1DataGrid>` tag so that it appears similar to the following:

   ```
   <c1:C1DataGrid x:Name="peopleDataGrid" AutoGenerateColumns="True"
   CanUserAddRows="False"
   LoadedRowPresenter="peopleDataGrid_LoadedRowPresenter">
   ```

   This markup adds an event handler – you'll add code for the event handler in the next steps.
6. In the Solution Explorer, expand the **MainPage.xaml** node and double-click the **MainPage.xaml.cs** or **MainPage.xaml.vb** file to open it in the Code Editor.
7. Add the following import statements at the top of the file:

   - Visual Basic

     ```
     Imports System.Runtime.Serialization
     Imports System.Collections.ObjectModel
     Imports System.ServiceModel
     Imports C1.Silverlight
     Imports C1.Silverlight.DataGrid
     Imports StealthPaging.DataService ' Change this if the name of your
     project is different.
     ```

   - C#

     ```
     using System.Runtime.Serialization;
     using System.Collections.ObjectModel;
     using System.ServiceModel;
     using C1.Silverlight;
     using C1.Silverlight.DataGrid;
     using StealthPaging.DataService; // Change this if the name of your
     project is different.
     ```

8. Add the following variables to the **MainPage** class:

   - Visual Basic

     ```
     Dim _startRow As Integer = 0
     ```

```
Dim _pageSize As Integer = 20
Dim _people As New ObservableCollection(Of ServerPerson)()
Dim _loading As Boolean
```

- C#

```
int _startRow = 0;
int _pageSize = 20;
ObservableCollection<ServerPerson> _people = new
ObservableCollection<ServerPerson>();
bool _loading;
```

9. Add code to the **MainPage** constructor so it appears like the following:

- Visual Basic

```
Public Sub New()
   InitializeComponent()
   AddHandler peopleDataGrid.LoadedRowPresenter, AddressOf
peopleDataGrid_LoadedRowPresenter
   peopleDataGrid.ItemsSource = _people
   GetData(_startRow, _pageSize)
End Sub
```

- C#

```
public MainPage()
{
    InitializeComponent();
    peopleDataGrid.LoadedRowPresenter += new
EventHandler<DataGridRowEventArgs>(peopleDataGrid_LoadedRowPresenter);
    peopleDataGrid.ItemsSource = _people;
    GetData(_startRow, _pageSize);
}
```

10. Add the **LoadedRowPresenter** event handler to your code under the **MainPage** constructor:

- Visual Basic

```
Private Sub peopleDataGrid_LoadedRowPresenter(ByVal sender As
System.Object, ByVal e As C1.Silverlight.DataGrid.DataGridRowEventArgs)
   If _loading OrElse _people.Count < _pageSize Then
         Return
   End If
   If _people.Count - 5 < e.Row.Index Then
         GetData(_startRow, _startRow + _pageSize)
   End If
End Sub
```

- C#

```
private void peopleDataGrid_LoadedRowPresenter(object sender,
C1.Silverlight.DataGrid.DataGridRowEventArgs e)
{
    if (_loading || _people.Count < _pageSize)
    {
        return;
    }
    if (_people.Count - 5 < e.Row.Index)
    {
        GetData(_startRow, _startRow + _pageSize);
    }
}
```

11. Add the following code to retrieve data from the server:

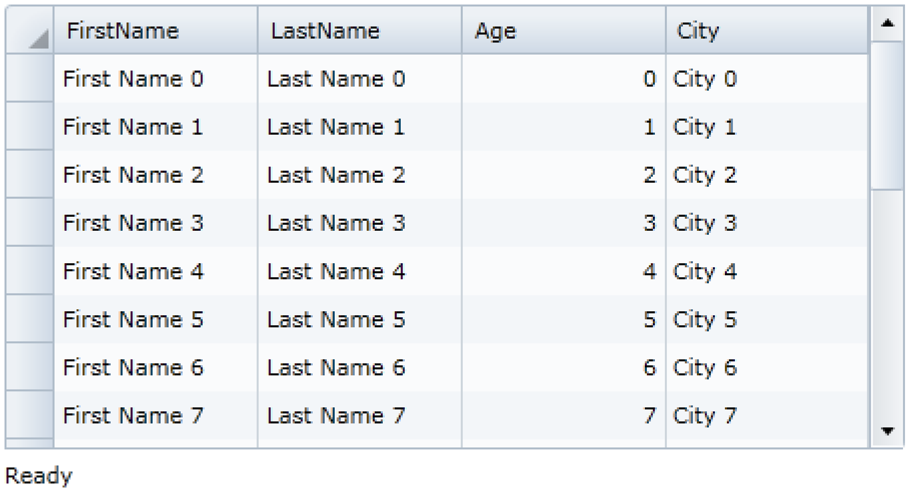
- Visual Basic

```
#Region "retrieve data from the server"
Private Sub GetData(startRow As Integer, endRow As Integer)
   UpdateState(True, startRow, endRow)
   ' Call web service
   Dim proxy = New DataWebServiceSoapClient(New BasicHttpBinding(), New
EndpointAddress(Extensions.GetAbsoluteUri("DataWebService.asmx")))
   AddHandler proxy.GetDataCompleted, AddressOf proxy_GetDataCompleted
   proxy.GetDataAsync(startRow, endRow)
End Sub
Private Sub proxy_GetDataCompleted(sender As Object, e As
GetDataCompletedEventArgs)
   If e.[Error] IsNot Nothing Then
         MessageBox.Show(e.[Error].Message, "Error Getting Data",
MessageBoxButton.OK)
         Return
   End If
   ' Data retrieved OK, add to observable collection
   _startRow += _pageSize
   For Each person As ServerPerson In e.Result
         _people.Add(person)
   Next
   UpdateState(False, 0, 0)
End Sub
' Sets loading status
' You could use a VisualState here too
Private Sub UpdateState(loading As Boolean, startRow As Integer, endRow
As Integer)
   If loading Then
         txtStatus.Text = String.Format("Retrieving rows {0} to
{1}...", startRow, endRow)
         Cursor = Cursors.Wait
         _loading = True
   Else
         _loading = False
         txtStatus.Text = "Ready"
         Cursor = Cursors.Arrow
   End If
End Sub
#End Region
```

- C#

```
#region retrieve data from the server
private void GetData(int startRow, int endRow)
{
    UpdateState(true, startRow, endRow);
    // Call Web service
    var proxy = new DataWebServiceSoapClient(new BasicHttpBinding(),
new EndpointAddress(Extensions.GetAbsoluteUri("DataWebService.asmx")));
    proxy.GetDataCompleted += new
EventHandler<GetDataCompletedEventArgs>(proxy_GetDataCompleted);
    proxy.GetDataAsync(startRow, endRow);
}
void proxy_GetDataCompleted(object sender, GetDataCompletedEventArgs e)
{
    if (null != e.Error)
    {
        MessageBox.Show(e.Error.Message, "Error Getting Data",
MessageBoxButton.OK);
        return;
    }
    // Data retrieved OK, add to observable collection
    _startRow += _pageSize;
    foreach (ServerPerson person in e.Result)
    {
        _people.Add(person);
    }
    UpdateState(false, 0, 0);
}
// Sets loading status
// You could use a VisualState here too
private void UpdateState(bool loading, int startRow, int endRow)
{
    if (loading)
    {
        txtStatus.Text = string.Format("Retrieving rows {0} to {1}...",
startRow, endRow);
        Cursor = Cursors.Wait;
        _loading = true;
    }
    else
    {
        _loading = false;
        txtStatus.Text = "Ready";
        Cursor = Cursors.Arrow;
    }
}
#endregion
```

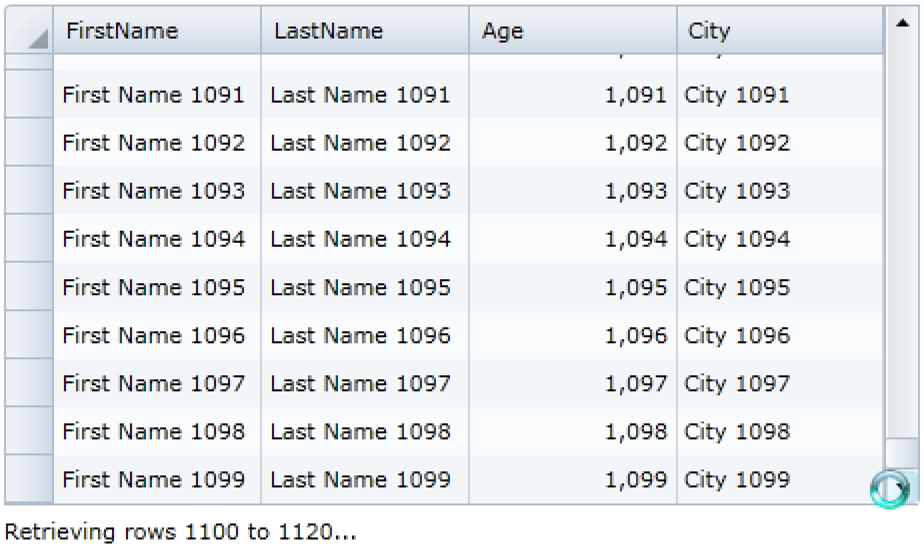
12. Run your application and observe that the grid appears bound to a data source:



FirstName	LastName	Age	City
First Name 0	Last Name 0	0	City 0
First Name 1	Last Name 1	1	City 1
First Name 2	Last Name 2	2	City 2
First Name 3	Last Name 3	3	City 3
First Name 4	Last Name 4	4	City 4
First Name 5	Last Name 5	5	City 5
First Name 6	Last Name 6	6	City 6
First Name 7	Last Name 7	7	City 7

Ready

13. Run your application and observe that as you scroll through the grid more rows appear in the grid:



FirstName	LastName	Age	City
First Name 1091	Last Name 1091	1,091	City 1091
First Name 1092	Last Name 1092	1,092	City 1092
First Name 1093	Last Name 1093	1,093	City 1093
First Name 1094	Last Name 1094	1,094	City 1094
First Name 1095	Last Name 1095	1,095	City 1095
First Name 1096	Last Name 1096	1,096	City 1096
First Name 1097	Last Name 1097	1,097	City 1097
First Name 1098	Last Name 1098	1,098	City 1098
First Name 1099	Last Name 1099	1,099	City 1099

Retrieving rows 1100 to 1120...

Also note that the text below the grid indicates the rows being added as you scroll.

What You've Accomplished

Congratulations, you've completed this tutorial! In this tutorial you created a new Silverlight project, added a data source, and created a Web Service to bind the **C1DataGrid** control. You implemented stealth paging, so that when the grid is scrolled at run time, the grid pages through the grid instead, improving performance.

DataGrid for Silverlight Task-Based Help

The following task-based help topics assume that you are familiar with Visual Studio and Expression Blend and know how to use the C1DataGrid control in general. If you are unfamiliar with the **ComponentOne DataGrid for Silverlight** product, please see the DataGrid for Silverlight Quick Start (page 46) first.

Each topic in this section provides a solution for specific tasks using the **ComponentOne DataGrid for Silverlight** product. Most task-based help topics also assume that you have created a new WPF project and added a C1DataGrid control to the project – for information about creating the control, see Creating a DataGrid (page 144).

Creating a DataGrid

You can easily create a C1DataGrid control at design time in Expression Blend, in XAML, and in code. Note that if you create a C1DataGrid control as in the following steps, it will appear empty. You will need to bind the grid or populate it with data.

At Design Time in Blend

To create a C1DataGrid control in Blend, complete the following steps:

1. Navigate to the **Projects** window and right-click the **References** folder in the project files list. In the context menu choose **Add Reference**, locate and select the **C1.Silverlight.DataGrid.dll** assembly, and click **Open**.

The dialog box will close and the references will be added to your project and the controls will be available in the Asset Library.

2. In the Toolbox click on the **Assets** button (the double chevron icon) to open the **Assets** dialog box.
3. In the **Asset Library** dialog box, choose the **Controls** item in the left pane, and then click on the **C1DataGrid** icon in the right pane:

   The **C1DataGrid** icon will appear in the Toolbox under the **Assets** button.
4. Click once on the design area of the **UserControl** to select it. Unlike in Visual Studio, in Blend you can add WPF controls directly to the design surface as in the next step.
5. Double-click the **C1DataGrid** icon in the Toolbox to add the control to the panel. The C1DataGrid control will now exist in your application.
6. If you choose, can customize the control by selecting it and setting properties in the Properties window. For example, set the C1DataGrid control's **Name** property to "c1datagrid1" the **Height** property to "180", and the **Width** property to "250".

In XAML

To create a C1DataGrid control using XAML markup, complete the following steps:

1. In the Visual Studio Solution Explorer, right-click the **References** folder in the project files list. In the context menu choose **Add Reference**, select the **C1.WPF.DataGrid.dll** assembly, and click **OK**.
2. Add a XAML namespace to your project by adding `xmlns:c1="http://schemas.componentone.com/winfx/2006/xaml` to the initial `<UserControl>` tag. It will appear similar to the following:
```
<UserControl
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:c1="http://schemas.componentone.com/winfx/2006/xaml
x:Class="C1DataGrid.MainPage" Width="640" Height="480">
```
3. Add a `<c1:C1DataGrid>` tag to your project within the `<Grid>` tag to create a C1DataGrid control. The markup will appear similar to the following:
```
<Grid x:Name="LayoutRoot" Background="White">
    <c1:C1DataGrid Name="c1datagrid1" Height="180" Width="250" />
</Grid>
```
   This markup will create an empty C1DataGrid control named "c1datagrid1" and set the control's size.

In Code

To create a C1DataGrid control in code, complete the following steps:

1. In the Visual Studio Solution Explorer, right-click the **References** folder in the project files list. In the context menu choose **Add Reference**, select the **C1.Silverlight.dll** and **C1.Silverlight.DataGrid.dll** assemblies, and click **OK**.
2. Right-click within the **MainPage.xaml** window and select **View Code** to switch to Code view
3. Add the following import statements to the top of the page:
   - Visual Basic
   ```
   Imports C1.Silverlight.DataGrid
   ```
   - C#
   ```
   using C1.Silverlight.DataGrid;
   ```
4. Add code to the page's constructor to create the C1DataGrid control. It will look similar to the following:
   - Visual Basic
   ```
   Public Sub New()
   ```

```
    InitializeComponent()
    Dim c1datagrid1 As New C1DataGrid
    c1datagrid1.Height = 180
    c1datagrid1.Width = 250
    LayoutRoot.Children.Add(c1datagrid1)
End Sub
```

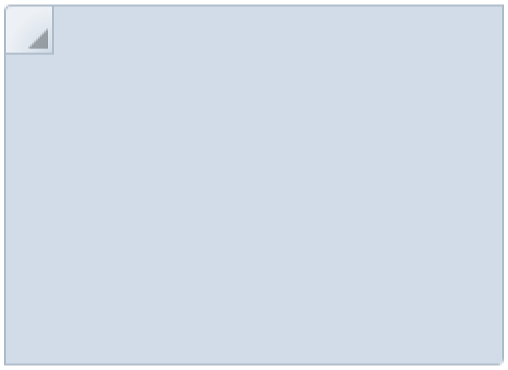
- C#

```
public MainPage()
{
    InitializeComponent();
    C1DataGrid c1datagrid1 = new C1DataGrid();
    c1datagrid1.Height = 180;
    c1datagrid1.Width = 250;
    LayoutRoot.Children.Add(c1datagrid1);
}
```

This code will create an empty C1DataGrid control named "c1datagrid1", set the control's size, and add the control to the page.

What You've Accomplished

Run your application and observe that you've created a C1DataGrid control.



Note that when you create a C1DataGrid control as in the above steps, it will appear empty. You can add items to the control that can be interacted with at run time.

Controlling Grid Interaction

The following task-based help topics detail how you can enhance your users' interaction with **DataGrid for Silverlight**. For example, you can allow users to filter, sort, reorder, delete, and edit the grid through code and XAML.

Enabling Grouping in the Grid

You can enable grouping and the grouping area of the grid so that users can group columns in your grid at run time to better organize information. For more information, see Grouping Columns (page 98). By default, user cannot group columns in the grid but you can enable this function by setting the CanUserGroup property to **True**.

At Design Time

To enable grouping, complete the following steps:

1. Click the C1DataGrid control once to select it.

2. Navigate to the Properties window and locate the CanUserGroup property.
3. Check the check box next to the CanUserGroup property.

In XAML

For example to enable grouping, add `CanUserGroup="True"` to the `< c1:C1DataGrid>` tag so that it appears similar to the following:

```
<c1:C1DataGrid Name="c1datagrid1" Height="180" Width="250"
CanUserGroup=True" />
```

In Code

For example, to enable grouping, add the following code to your project:

- Visual Basic

```
Me.C1DataGrid1.CanUserGroup = True
```

- C#

```
this.c1DataGrid1.CanUserGroup = true;
```

What You've Accomplished

Run the application and notice that the grouping area appears at the top of the grid. Note that you can also customize the visibility of the grouping area. For more information about the grouping area, see the Showing the Grouping Area (page 147) topic.

Showing the Grouping Area

By default grouping in the grid is disabled and the grouping area is not visible. For more information, see Grouping Columns (page 98). When the CanUserGroup property is set to **True** and grouping is enabled the grouping area is made visible. But if you choose you can show or hide the grouping area whether or not grouping is enabled. By default, the grouping area is not visible when grouping is not enabled but you can make the area visible by setting the ShowGroupingPanel property to **True**.

At Design Time

To show the grouping area, complete the following steps:

1. Click the C1DataGrid control once to select it.
2. Navigate to the Properties window and locate the ShowGroupingPanel property.
3. Check the check box next to the ShowGroupingPanel property.

In XAML

For example to show the grouping area, add `ShowGroupingPanel="True"` to the `< c1:C1DataGrid>` tag so that it appears similar to the following:

```
<c1:C1DataGrid Name="c1DataGrid1" Height="180" Width="250"
ShowGroupingPanel="True" />
```

In Code

For example, to show the grouping area, add the following code to your project:

- Visual Basic

```
Me.C1DataGrid1.ShowGroupingPanel = True
```

- C#

```
this.c1DataGrid1.ShowGroupingPanel = true;
```

What You've Accomplished

Run the application and notice that the grouping area appears at the top of the grid. Note that even if the grouping area is visible, grouping will not be enabled if the CanUserGroup property is **False**. For more information, see the Enabling Grouping in the Grid (page 146) topic.

Disabling Column Reordering

By default end users can easily reorder columns in the grid at run time. For more information, see Reordering Columns (page 95). If you choose, however, you can disable the column reordering feature by setting the CanUserReorderColumns property to **False**.

At Design Time

To disable column reordering, complete the following steps:

1. Click the C1DataGrid control once to select it.
2. Navigate to the Properties window and locate the CanUserReorderColumns property.
3. Clear the check box next to the CanUserReorderColumns property.

In XAML

For example to disable column reordering, add `CanUserReorderColumns="False"` to the `< c1:C1DataGrid>` tag so that it appears similar to the following:

```
<c1:C1DataGrid Name="c1datagrid1" Height="180" Width="250"
CanUserReorderColumns="False" />
```

In Code

For example, to disable column reordering, add the following code to your project:

- Visual Basic

```
Me.C1DataGrid1.CanUserReorderColumns = False
```

- C#

```
this.c1DataGrid1.CanUserReorderColumns = false;
```

What You've Accomplished

Run the application and observe that you can no longer reorder columns at run time by preforming a drag-and-drop operation. For more information about column reordering, see the Reordering Columns (page 95) topic.

Disabling Column and Row Resizing

By default end users can resize columns and rows in the grid at run time. For more information, see Resizing Columns and Rows (page 94). If you choose, however, you can disable the column and row resizing feature by setting the CanUserResizeColumns and CanUserResizeRows properties to **False**.

At Design Time

To disable column and row resizing, complete the following steps:

1. Click the C1DataGrid control once to select it.
2. Navigate to the Properties window and locate the CanUserResizeColumns property.
3. Clear the check box next to the CanUserResizeColumns property.
4. In the Properties window, locate the CanUserResizeRows property.
5. Clear the check box next to the CanUserResizeRows property.

In XAML

For example to disable column and row resizing, add `CanUserResizeColumns="False" CanUserResizeRows="False"` to the `< c1:C1DataGrid>` tag so that it appears similar to the following:

```
<c1:C1DataGrid Name="c1datagrid1" Height="180" Width="250"
CanUserResizeColumns="False" CanUserResizeRows="False"/>
```

In Code

For example, to disable column and row resizing, add the following code to your project:

- Visual Basic

```
Me.C1DataGrid1.CanUserResizeColumns = False
Me.C1DataGrid1.CanUserResizeRows = False
```

- C#

```
this.c1DataGrid1.CanUserResizeColumns = false;
this.c1DataGrid1.CanUserResizeRows = false;
```

What You've Accomplished

Run the application and observe that you can no longer resize columns or rows at run time by preforming a drag-and-drop operation. For more information about column reordering, see the Resizing Columns and Rows (page 94) topic.

Disabling Column Filtering

By default end users can filter columns in the grid at run time. For more information, see Filtering Columns (page 95). If you choose, however, you can disable the column filtering feature by setting the CanUserFilter property to **False**.

At Design Time

To disable column filtering, complete the following steps:

1. Click the C1DataGrid control once to select it.
2. Navigate to the Properties window and locate the CanUserFilter property.
3. Clear the check box next to the CanUserFilter property.

In XAML

For example to disable column filtering, add `CanUserFilter="False"` to the `< c1:C1DataGrid>` tag so that it appears similar to the following:

```
<c1:C1DataGrid Name="c1datagrid1" Height="180" Width="250"
CanUserFilter="False" />
```

In Code

For example, to disable column filtering, add the following code to your project:

- Visual Basic

```
Me.C1DataGrid1.CanUserFilter = False
```

- C#

```
this.c1DataGrid1.CanUserFilter = false;
```

What You've Accomplished

Run the application and observe that you can no longer filter columns at run time; the drop-down arrow to display the filter box is no longer visible at run time. For more information about column filtering, see the Filtering Columns (page 95) topic.

Disabling Column Sorting

By default end users can sort columns in the grid at run time. For more information, see Sorting Columns (page 97). If you choose, however, you can disable the column sorting feature by setting the CanUserSort property to **False**.

At Design Time

To disable column sorting, complete the following steps:

1. Click the C1DataGrid control once to select it.
2. Navigate to the Properties window and locate the CanUserSort property.
3. Clear the check box next to the CanUserSort property.

In XAML

For example to disable column sorting, add `CanUserSort="False"` to the `< c1:C1DataGrid>` tag so that it appears similar to the following:

```
<c1:C1DataGrid Name="c1datagrid1" Height="180" Width="250"
CanUserSort="False" />
```

In Code

For example, to disable column sorting, add the following code to your project:

- Visual Basic
```
Me.C1DataGrid1.CanUserSort = False
```
- C#
```
this.c1DataGrid1.CanUserSort = false;
```

What You've Accomplished

Run the application and observe that you can no longer sort columns at run time. Clicking on a column's header at run time will not sort the grid and the sort indicator is not visible in the column header. For more information about column sorting, see the Sorting Columns (page 97) topic.

Enabling Column Freezing

You may want to freeze columns in the grid at run time so that they are always visible even when the grid is scrolled horizontally. For more information, see Freezing Columns (page 100). This feature is not enabled by default, but if you choose you can enable the column freezing feature by setting the CanUserFreezeColumns property to **Left**.

At Design Time

To enable column freezing, complete the following steps:

1. Click the C1DataGrid control once to select it.
2. Navigate to the Properties window and locate the CanUserFreezeColumns property.
3. Click the drop-down arrow next to the CanUserFreezeColumns property and select **Left**.

In XAML

For example to enable column freezing, add `CanUserFreezeColumns="Left"` to the `<c1:C1DataGrid>` tag so that it appears similar to the following:

```
<c1:C1DataGrid Name="c1datagrid1" Height="180" Width="250"
CanUserFreezeColumns="Left" />
```

In Code

For example, to enable column freezing, add the following code to your project:

- Visual Basic
```
Me.C1DataGrid1.CanUserFreezeColumns = DataGridColumnFreezing.Left
```
- C#
```
this.c1DataGrid1.CanUserFreezeColumns = DataGridColumnFreezing.Left;
```

What You've Accomplished

Run the application and observe that the freeze bar is visible at run time. The freeze bar can be moved to select which columns to freeze; columns to the left of the bar will be frozen so that they are always visible even when the grid is scrolled horizontally. For more information about column freezing, see the Freezing Columns (page 100) topic.

Freezing Grid Rows

You may want to freeze the top or bottom rows in the grid at so that they are always visible even when the grid is scrolled vertically at run time. This feature is not enabled by default, but if you choose you can enable the row freezing feature by setting the FrozenTopRowsCount and FrozenBottomRowsCount properties.

At Design Time

To freeze the top and bottom two rows, complete the following steps:

1. Click the C1DataGrid control once to select it and navigate to the Properties window.
2. In the Properties window, locate the FrozenTopRowsCount property, click in the text box next to the property, and enter "2" to set the number of top tows that will be frozen.
3. Locate the FrozenBottomRowsCount property, click in the text box next to the property, and enter "2" to set the number of bottom rows that will be frozen.

In XAML

For example to freeze the top and bottom two rows, add `FrozenTopRowsCount="2" FrozenBottomRowsCount="2"` to the `<c1:C1DataGrid>` tag so that it appears similar to the following:

```
<c1:C1DataGrid Name="c1datagrid1" Height="180" Width="250"
FrozenTopRowsCount="2" FrozenBottomRowsCount="2" />
```

In Code

For example, to freeze the top and bottom two rows, add the following code to your project:

- Visual Basic

```
Me.C1DataGrid1.FrozenTopRowsCount = True
Me.C1DataGrid1.FrozenBottomRowsCount = True
```

- C#

```
this.c1DataGrid1.FrozenTopRowsCount = true;
this.c1DataGrid1.FrozenBottomRowsCount = true;
```

What You've Accomplished

Run the application and observe that the two top and bottom rows are frozen. Scroll the grid vertically and notice that the top two an bottom two rows do not scroll and are locked in place. By default the Add New row appears as the last row in the grid and so will be one of the frozen rows.

Disabling Cell Editing

By default end users edit content in the grid at run time. For more information, see Editing Cells (page 101). If you choose, however, you can disable the cell editing feature by setting the CanUserEditRows property to **False**.

At Design Time

To disable cell editing, complete the following steps:

1. Click the C1DataGrid control once to select it.
2. Navigate to the Properties window and locate the CanUserEditRows property.
3. Clear the check box next to the CanUserEditRows property.

In XAML

For example to disable cell editing, add `CanUserEditRows="False"` to the `< c1:C1DataGrid>` tag so that it appears similar to the following:

```
<c1:C1DataGrid Name="c1datagrid1" Height="180" Width="250"
CanUserEditRows="False" />
```

In Code

For example, to disable cell editing, add the following code to your project:

- Visual Basic

```
Me.C1DataGrid1.CanUserEditRows = False
```

- C#

```
this.c1DataGrid1.CanUserEditRows = false;
```

What You've Accomplished

Run the application and double-click a cell; observe that the cell does not move into edit mode and you can no longer edit grid content at run time. For more information about cell editing, see the Editing Cells (page 101) topic.

Disabling Adding Rows

By default end users add new rows and content to the grid at run time. A new row bar appears at the bottom of the grid, users can enter text in the bar to add new content to the grid. For more information, see Adding Rows to the Grid (page 102). If you choose, however, you can disable the new row bar feature by setting the CanUserAddRows property to **False**.

At Design Time

To disable adding rows, complete the following steps:

1. Click the C1DataGrid control once to select it.
2. Navigate to the Properties window and locate the CanUserAddRows property.
3. Clear the check box next to the CanUserAddRows property.

In XAML

For example to disable adding rows, add `CanUserEditRows="False"` to the `< c1:C1DataGrid>` tag so that it appears similar to the following:

```
<c1:C1DataGrid Name="c1datagrid1" Height="180" Width="250"
CanUserAddRows="False" />
```

In Code

For example, to disable adding rows, add the following code to your project:

- Visual Basic

```
Me.C1DataGrid1.CanUserAddRows = False
```

- C#

```
this.c1DataGrid1.CanUserAddRows = false;
```

What You've Accomplished

Run the application and scroll to the end of the grid, if needed. Observe that the new row bar no longer appears in the grid and that users can no longer add new rows and content to the grid. For more information about cell editing, see the Adding Rows to the Grid (page 102) topic.

Disabling Row Details Toggling

When the grid includes a child grid or you've created a master-detail grid, by default the row details can be toggled so that they are visible or collapsed. If you choose, however, you can disable the toggling the details row feature by setting the CanUserToggleDetails property to **False**. Note that you will need to have a grid with row details to view the change in this example.

At Design Time

To disable toggling row details, complete the following steps:

1. Click the C1DataGrid control once to select it.
2. Navigate to the Properties window and locate the CanUserToggleDetails property.
3. Clear the check box next to the CanUserToggleDetails property.

In XAML

For example to disable toggling row details, add `CanUserToggleDetails="False"` to the `< c1:C1DataGrid>` tag so that it appears similar to the following:

```
<c1:C1DataGrid Name="c1datagrid1" Height="180" Width="250"
CanUserToggleDetails="False" />
```

In Code

For example, to disable toggling row details, add the following code to your project:

- Visual Basic
```
Me.C1DataGrid1.CanUserToggleDetails = False
```
- C#
```
this.c1DataGrid1.CanUserToggleDetails = false;
```

What You've Accomplished

Run the application and observe that you can no longer toggle the row details in the grid at run time. The arrow icon in the row header that indicates that row details can be toggled is no longer visible so toggling rows is not an option.

Customizing Grid Appearance

The following task-based help topics detail how you can customize **DataGrid for Silverlight** by changing the grid's appearance. **DataGrid for Silverlight** includes several appearance options that incorporate ComponentOne's unique ClearStyle technology. For example, you can change the background color of the grid or the alternating row background. Note for more information about ClearStyle technology, see the C1DataGrid ClearStyle (page 87) topic. The follow topics also detail changing the layout of the grid, including how to set the location of the header and add new row bar.

Changing the Grid's Background and Foreground Color

ComponentOne DataGrid for Silverlight includes ComponentOne's unique ClearStyle technology that enables you to change the entire appearance of the grid simply and flawlessly. The following steps will detail how to set the **C1DataGrid.Background** property to completely change the appearance of the grid. For more details about ComponentOne's ClearStyle technology, see the C1DataGrid ClearStyle (page 87) topic.

At Design Time

To change the grid's foreground and background color so that it appears green, complete the following steps:

1. Click the C1DataGrid control once to select it.
2. Navigate to the Properties window and click the drop-down arrow next to the **Background** property.
3. Click the drop-down arrow in the box the hex code appears in, and choose **Green**.
4. Navigate to the Properties window and click the drop-down arrow next to the **Foreground** property.
5. Click the drop-down arrow in the box the hex code appears in, and choose **White**.

In XAML

For example to change the grid's foreground and background color so that it appears green, add `Background="Green" Foreground="White"` to the `<c1:C1DataGrid>` tag so that it appears similar to the following:

```
<c1:C1DataGrid Name="c1datagrid1" Height="180" Width="250"
Background="Green" Foreground="White" />
```

In Code

For example, to change the grid's foreground and background color so that it appears green, add the following code to your project:

- Visual Basic

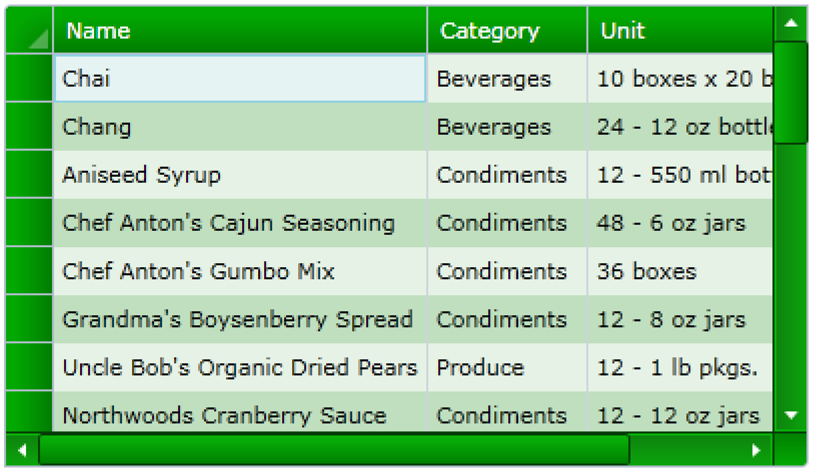
```
Me.C1DataGrid1.Background = New
System.Windows.Media.SolidColorBrush(Colors.Green)
Me.C1DataGrid1.ForeGround = New
System.Windows.Media.SolidColorBrush(Colors.White)
```

- C#

```
this.c1DataGrid1.Background = new System.Windows.Media.
SolidColorBrush(Colors.Green);
this.c1DataGrid1.Foreground = new System.Windows.Media.
SolidColorBrush(Colors.White);
```

What You've Accomplished

Run the application and observe that the grid now appears green with white text in the grid header.



Name	Category	Unit
Chai	Beverages	10 boxes x 20 b
Chang	Beverages	24 - 12 oz bottl
Aniseed Syrup	Condiments	12 - 550 ml bot
Chef Anton's Cajun Seasoning	Condiments	48 - 6 oz jars
Chef Anton's Gumbo Mix	Condiments	36 boxes
Grandma's Boysenberry Spread	Condiments	12 - 8 oz jars
Uncle Bob's Organic Dried Pears	Produce	12 - 1 lb pkgs.
Northwoods Cranberry Sauce	Condiments	12 - 12 oz jars

Note that with the C1DataGrid control's ClearStyle technology, the color of the grid, the grid's scrollbars, and the alternating row background of the grid all changed to reflect the green background. Highlight an item in the grid and notice the mouse hover style did not change; you can customize these styles as well if you choose. See Changing the Grid's Mouse Hover Style (page 156) for more details.

Removing the Grid's Alternating Row Colors

ComponentOne DataGrid for Silverlight appears with alternating row colors by default. Alternating row colors are when alternate lines appear in a different color than the base color of the grid. This is helpful so that rows are easier to follow across the grid, but if you choose you can make the appearance of the grid uniform by removing the alternating row colors.

At Design Time

To remove alternating row colors and set it so all rows appear white, complete the following steps:

1. Click the C1DataGrid control once to select it.
2. Navigate to the Properties window and click the drop-down arrow next to the **RowBackground** property.
3. Click the drop-down arrow in the box the hex code appears in, and choose **White**.
4. Navigate to the Properties window and click the drop-down arrow next to the **AlternatingRowBackground** property.
5. Click the drop-down arrow in the box the hex code appears in, and choose **White**.

In XAML

To remove alternating row colors and set it so all rows appear white, add `RowBackground="White" AlternatingRowBackground="White"` to the `<c1:C1DataGrid>` tag so that it appears similar to the following:

```
<c1:C1DataGrid Name="c1datagrid1" Height="180" Width="250"
RowBackground="White" AlternatingRowBackground="White" />
```

In Code

To remove alternating row colors and set it so all rows appear white, add the following code to your project:

- Visual Basic

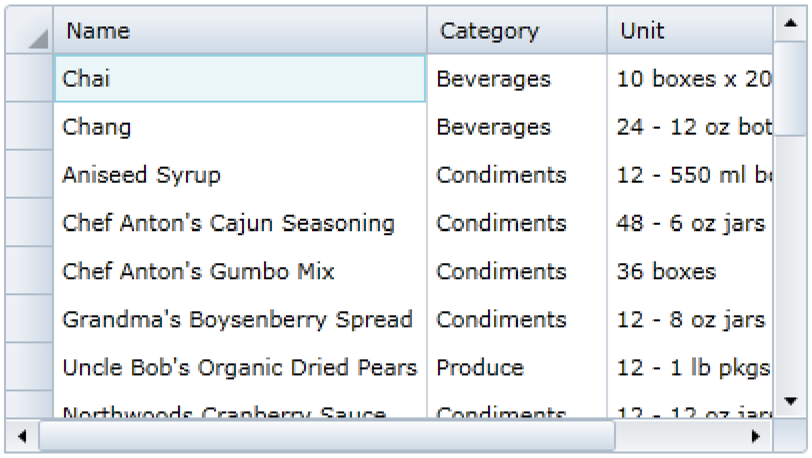
```
Me.C1DataGrid1.RowBackground = New
System.Windows.Media.SolidColorBrush(Colors.White)
Me.C1DataGrid1.AlternatingRowBackground = New
System.Windows.Media.SolidColorBrush(Colors.White)
```

- C#

```
this.c1DataGrid1.RowBackground = new System.Windows.Media.
SolidColorBrush(Colors.White);
this.c1DataGrid1.AlternatingRowBackground = new System.Windows.Media.
SolidColorBrush(Colors.White);
```

What You've Accomplished

Run the application and observe that all rows in the grid now appear white.



Name	Category	Unit
Chai	Beverages	10 boxes x 20
Chang	Beverages	24 - 12 oz bot
Aniseed Syrup	Condiments	12 - 550 ml b
Chef Anton's Cajun Seasoning	Condiments	48 - 6 oz jars
Chef Anton's Gumbo Mix	Condiments	36 boxes
Grandma's Boysenberry Spread	Condiments	12 - 8 oz jars
Uncle Bob's Organic Dried Pears	Produce	12 - 1 lb pkgs
Northwoods Cranberry Sauce	Condiments	12 - 12 oz jar

Changing the Grid's Mouse Hover Style

By default, columns and rows that are moused over appear in a different color to indicate to users what area of the grid they are interacting with. If you choose you can customize the appearance of cells that are moused over. For example, you may want to highlight these cells even more or remove this effect.

At Design Time

To set the mouse over effect to yellow, complete the following steps:

1. Click the C1DataGrid control once to select it.
2. Navigate to the Properties window and click the drop-down arrow next to the **MouseOverBrush** property.
3. Click the drop-down arrow in the box the hex code appears in, and choose **Yellow**.

In XAML

To set the mouse over effect to yellow, add `MouseOverBrush="Yellow"` to the `<c1:C1DataGrid>` tag so that it appears similar to the following:

```
<c1:C1DataGrid Name="c1datagrid1" Height="180" Width="250"
MouseOverBrush="Yellow" />
```

In Code

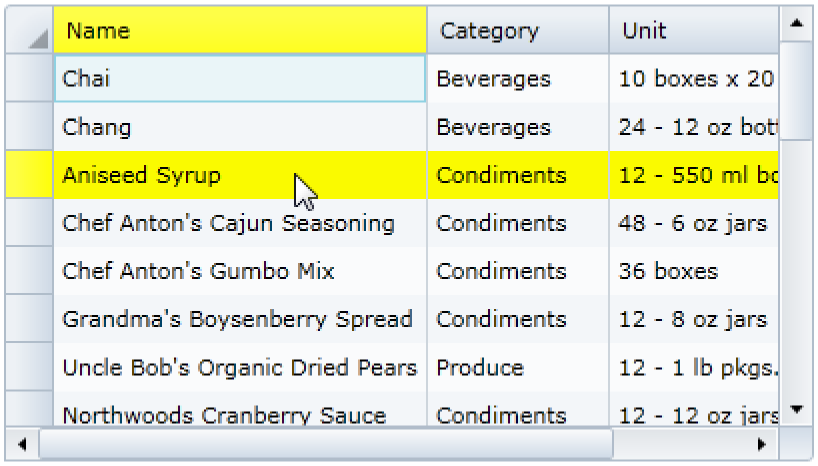
To set the mouse over effect to yellow, add the following code to your project:

- Visual Basic
```
Me.c1datagrid1.MouseOverBrush = New
System.Windows.Media.SolidColorBrush(Colors.Yellow)
```

- C#
```
this.c1datagrid1.MouseOverBrush = new
System.Windows.Media.SolidColorBrush(Colors.Yellow);
```

What You've Accomplished

Run the application and observe that all highlighted rows and columns in the grid now appear yellow.



Name	Category	Unit
Chai	Beverages	10 boxes x 20
Chang	Beverages	24 - 12 oz bott
Aniseed Syrup	Condiments	12 - 550 ml bo
Chef Anton's Cajun Seasoning	Condiments	48 - 6 oz jars
Chef Anton's Gumbo Mix	Condiments	36 boxes
Grandma's Boysenberry Spread	Condiments	12 - 8 oz jars
Uncle Bob's Organic Dried Pears	Produce	12 - 1 lb pkgs.
Northwoods Cranberry Sauce	Condiments	12 - 12 oz jars

Changing the Grid's Font Style

You may want to update the font style that appears in **DataGrid for Silverlight** when the control is run. For example, you may want to change the style of the grid, an element of which is the font style, to match your application's appearance.

At Design Time

To change the font style, complete the following steps:

1. Click the C1DataGrid control once to select it.
2. Navigate to the Properties window and click the drop-down arrow next to the **FontFamily** property and choose **Times New Roman**.

3. Navigate to the Properties window and click the drop-down arrow next to the **FontSize** property and choose **10**.

In XAML

To change the font style, add `FontFamily="Times New Roman" FontSize="10"` to the `<c1:C1DataGrid>` tag so that it appears similar to the following:

```
<c1:C1DataGrid Name="c1datagrid1" Height="180" Width="250"
FontFamily="Times New Roman" FontSize="10" />
```

In Code

To remove alternating row colors and set it so all rows appear white, add the following code to your project:

- Visual Basic

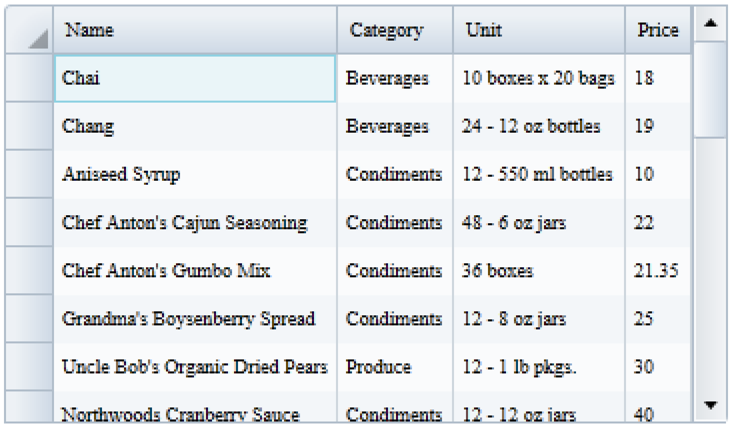
```
Me.c1datagrid1.FontFamily = New FontFamily("Times New Roman")
Me.c1datagrid1.FontSize = 10
```

- C#

```
this.c1datagrid1.FontFamily = new FontFamily("Times New Roman");
this.c1datagrid1.FontSize = 10;
```

What You've Accomplished

Run the application and observe that all rows in the grid appear in the Times New Roman font.



Name	Category	Unit	Price
Chai	Beverages	10 boxes x 20 bags	18
Chang	Beverages	24 - 12 oz bottles	19
Aniseed Syrup	Condiments	12 - 550 ml bottles	10
Chef Anton's Cajun Seasoning	Condiments	48 - 6 oz jars	22
Chef Anton's Gumbo Mix	Condiments	36 boxes	21.35
Grandma's Boysenberry Spread	Condiments	12 - 8 oz jars	25
Uncle Bob's Organic Dried Pears	Produce	12 - 1 lb pkgs.	30
Northwoods Cranberry Sauce	Condiments	12 - 12 oz jars	40